MODELING ATOMIC DEFECTS IN A TWO-DIMENSIONAL LENNARD-JONES LATTICE USING MOLECULAR DYNAMICS SIMULATIONS

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Andrew James Tuesday

May, 2012

MODELING ATOMIC DEFECTS IN A TWO-DIMENSIONAL LENNARD-JONES LATTICE USING MOLECULAR DYNAMICS SIMULATIONS

Andrew James Tuesday

Thesis

ABSTRACT

The word “nanotechnology” refers to manipulation and processing of materials at an atomistic or molecular level. One of the “hot” areas in nanotechnology is the study of fullerenes, in particular carbon nanotubes (CNTs). CNTs are among potentially promising materials in electronic applications due to their remarkable mechanical (high Young’s modulus and high tensile strength) and electronic properties. The building block of a CNT is a graphene sheet, which is essentially a single-layered lattice of covalently bonded carbon atoms. Experimental evidence has shown the existence of lattice defects in graphene sheets. Because these defects may degrade the unique properties of CNTs, the study of atomic defects in graphene sheets has become increasingly important in nanotechnology research. In this thesis, a set of two-dimensional MD simulations, in which atoms interact via the Lennard-Jones forces, is developed and utilized to model the behavior of three types of lattice defects: vacancies, dislocations, and interstitials. We conjecture that our observations provide insight into the dynamics of an imperfect lattice of graphene.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Graphene Sheets

Nanotechnology is a field of research that studies materials at the nanoscale, focusing on their processing and manipulation. One of the most active areas in nanotechnology is the study of carbon nanotubes (CNTs) [3], which have attracted attention in part because of their remarkable electronic and mechanical properties. For example, CNTs have been shown to have a high Young's modulus and a high tensile strength [4]. CNTs are categorized into two general types: single-walled nanotubes (SWNT) and multi-walled nanotubes (MWNT). Conceptually, a SWNT can be thought of as a rolled graphene sheet— a single layer of carbon atoms— that forms a hollow, seamless tube with a diameter of approximately 1 nanometer [5]. A MWNT is similar, but it consists of multiple concentric rolled graphene sheets. Because CNTs are composed from graphene sheets, studying the properties of graphene is instrumental to the understanding of the properties of CNTs, both single-walled and multi-walled.

As stated in the last paragraph, a single layer graphene sheet consists of a monolayer of covalently bonded carbon atoms. These atoms are arranged periodically and uniformly in six-atom hexagonal rings that form a hexagonal lattice. The distance between each pair of bonded atoms in equilibrium is approximately 0.142 nm.

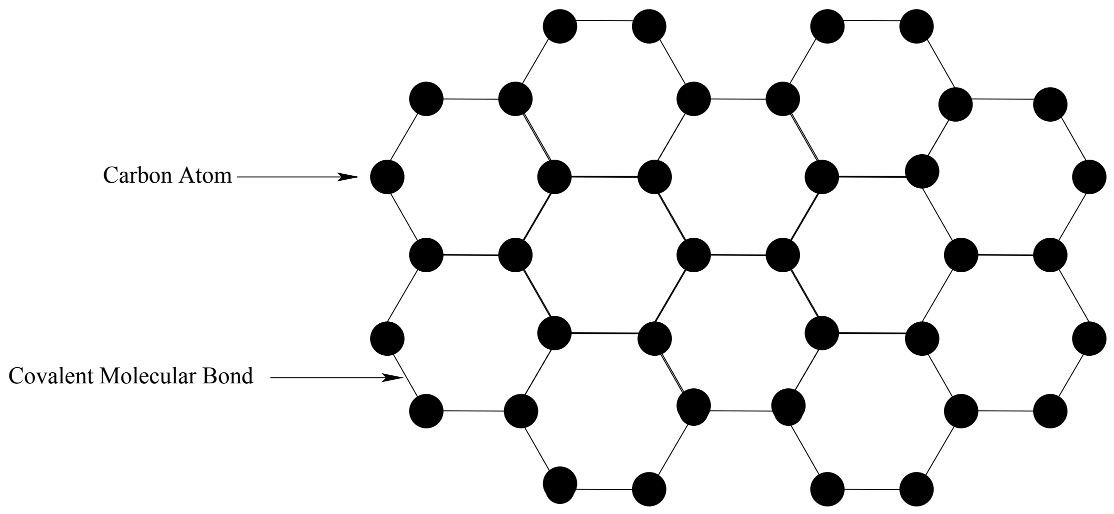


Figure 1.1: Schematic of the graphene sheet structure

Like carbon nanotubes, graphene exhibits unique mechanical and electronic properties. For example, graphene is a zero-gap semiconductor, meaning it displays properties which are characteristic of both semiconductors and metals [6]. Also, at room temperature, graphene exhibits high electron mobility, as well as high thermal conductivity. These characteristics make graphene an ideal material for many electronics applications [3].

One specific application is in transistors, because the properties of graphene may allow for transistors to become faster, smaller, and more efficient by consuming less energy than silicon-based devices. Graphene can also be used to enhance energy storage efficiency in batteries or ultra capacitors because it has a higher surface-to-volume ratio than other metals and semiconductors, as well as a higher conductivity, especially at room temperature [7]. In other studies, graphene has been used to produce DNA sensors, as well as bacterium bio devices. This is possible because of the high Young's modulus in graphene [8]. Due to its high tensile strength and flexibility, graphene can also be used in atomic force microscopy (AFM), a type of microscopy with very high resolution, on the order of fractions of a nanometer. This device requires a microscopic tip that is both flexible and strong, making graphene or CNTs an ideal material to scan surfaces for higher resolution AFM images [9]. More recently, CNTs have been found to be useful in drug delivery systems, because functionalized CNTs have low toxicity and are not immunogenic. This allows CNTs to be used for transporting therapeutic molecules throughout the body to cells and organs [10].

1.2 Modeling Graphene

Numerical or computational modeling of graphene can assist in the understanding of its properties and mechanical behavior. The mechanical properties of graphene are generally investigated using two different modeling techniques: continuum modeling and atomistic simulations [11].

Continuum modeling is an analytical approach based on theory of elasticity. The main difficulty, and most important aspect in this approach, is replacing the lattice structure of graphene with a continuum media [11]. Though this method saves computational expense and yields quicker results, there is potential for larger errors in predicted mechanical properties. For example, Hemmasizadeh et al., [12], present a method to develop an equivalent continuum model for a single-layered graphene sheet to independently obtain the effective Young's modulus and mechanical thickness of the sheet wall. While the proposed model succeeds in computing both of these values, the problem is reduced to first order from asymptotic expansion, resulting in large approximation errors, some of which are greater than 30 percent [12].

The second approach, atomistic simulations, includes a broad range of numerical and computational techniques. While these techniques may be computationally expensive and limit the size of the systems of atoms that can be modeled, they have potential for yielding more accurate results than continuum modeling. Two types of atomistic simulations are ab initio and molecular dynamics (MD). Ab initio modeling involves solving the Schrödinger equation directly for a given system of atoms. However, as suggested in [3], the standard Schrödinger equation might not be sufficient for modeling the unconventional properties of graphene [3]. The study presented in this thesis focuses on a less accurate but simpler MD approach, which utilizes Newton's second law to track the position and velocity of each individual atom throughout a lattice of atoms [5].

MD simulation of graphene is conducted, for example, by Bu et al. [13], where the mechanical behavior of graphene nanoribbons (GNRs)— thin strips of graphene, or rolled out SWNTs— is investigated. In [13], MD is used to show that under tensile loads, the elastic behavior of GNRs is nonlinear. Also, it is noted that the width of the GNR (or circumference of the CNT before it is rolled out) has a small effect on the Young's modulus [13]. Ni et al. employ MD along with a Verlet algorithm to investigate the anisotropic mechanical properties of graphene sheets [14]. In another study Bao et al., [15], use MD coupled with the Verlet algorithm to measure the Young's modulus of single-walled zigzag, armchair, and chiral CNTs. Using MD, the short-range interactions between atoms in [15] are modeled by a second-order empirical bond order, while long-range interactions are modeled by the Lennard-Jones potential. The resulting Young's moduli for each type are then compared to their experimental values and found to be within 1 percent accuracy [15]. This study along with others supports the feasibility of using MD for simulating graphene and CNTs.

1.3 Molecular Dynamics Simulations

In this study, MD simulation is used to model a possibly imperfect lattice of atoms interacting via the Lennard-Jones potential. Molecular dynamics utilizes a numerical step-by-step solution of the classical equations of motion based on Newton's second law, in order to simulate the motion of atoms over time. For a system of interacting atoms these equations may be written as

$$m\ddot{\mathbf{x}}_i = \mathbf{f}_i, \qquad \mathbf{f}_i = -\nabla_i U, \tag{1.1}$$

where $\mathbf{f}_i$ is the force acting on an individual atom i, $\mathbf{x}_i$ is the position of an atom i, m is the mass of atoms, and U is the potential energy of interactions between atoms in the system. The commonly used potential function in MD is the Lennard-Jones

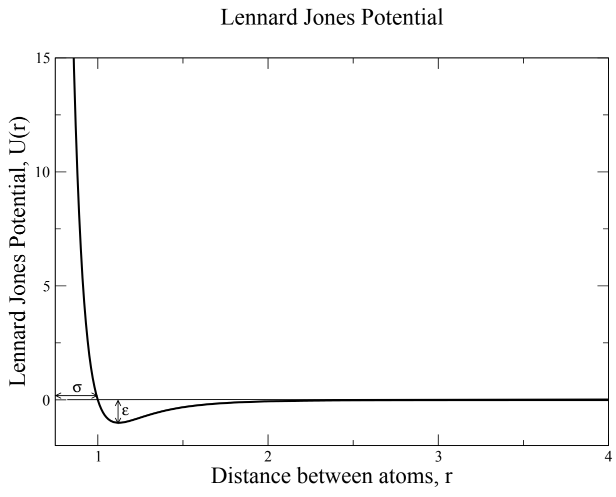


Figure 1.2: Graph of the Lennard-Jones potential function, U

potential [16], which is given by

$$U(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right], \tag{1.2}$$

so that

$$f(r) = -\frac{\partial}{\partial r}U(r) = 24\epsilon\left[2\frac{\sigma^{12}}{r^{13}} - \frac{\sigma^{6}}{r^{7}}\right], \tag{1.3}$$

where $f(r)$ is the magnitude of the force between two atoms separated by the distance r. Further, ϵ is the well depth describing the strength of interaction, and σ represents the distance between two atoms at which the potential is equal to zero. These are shown in the Lennard-Jones potential graph in Figure 1.2.

If the distance between two atoms is small, or less than the equilibrium separation, the Lennard-Jones force is strongly repulsive; at intermediate distances it is weakly attractive, and as r increases, the force decays rapidly (Figure 1.3.b).

By setting the force $f = 0$ in (1.3) and solving for r, the equilibrium distance r_{eq} is found to be

$$r_{eq} = 2^{\frac{1}{6}}\sigma. \tag{1.4}$$

If the distance between two atoms is less than r_{eq}, the Lennard-Jones force is repulsive. If the distance is greater than r_{eq}, the force is attractive (Figure 1.3). Note that when a chain or lattice of atoms is in equilibrium, the adjacent atoms are separated by a distance slightly less than r_{eq} due to second neighbor interactions.

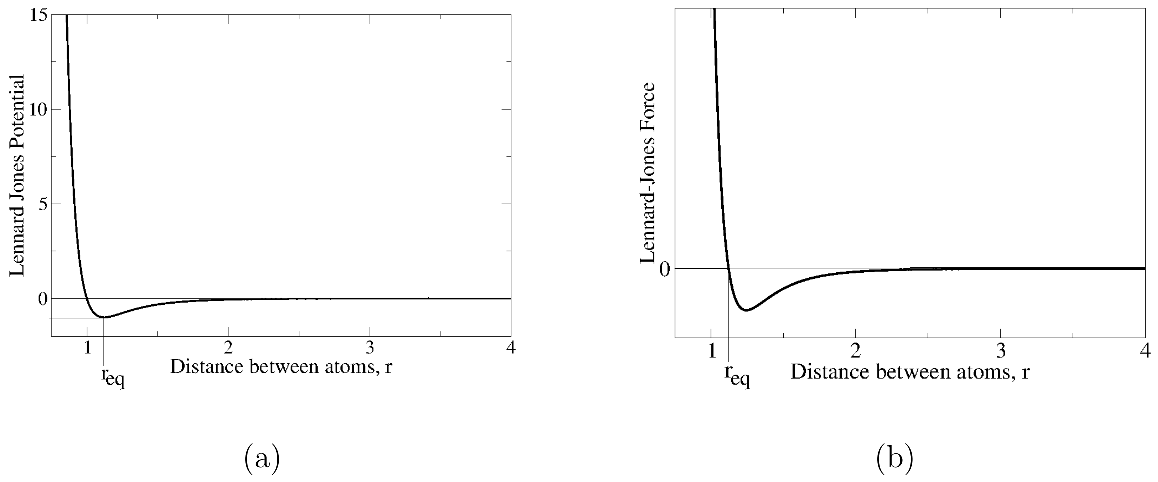


Figure 1.3: Graphs of the (a) Lennard-Jones potential and (b) Lennard-Jones force

The force exerted on atom i by the atom j is the gradient of U, given by

$$\mathbf{f}_{ij} = -\nabla_{\mathbf{r}_{ij}} U(r_{ij}) = -f(r_{ij})\frac{\mathbf{r}_{ij}}{r_{ij}} = 24\epsilon\left[\frac{\sigma^6}{{r_{ij}}^7} - 2\frac{\sigma^{12}}{{r_{ij}}^{13}}\right]\frac{\mathbf{r}_{ij}}{r_{ij}} \quad , \tag{1.5}$$

where $\mathbf{r}_{ij}$ is a vector from atom i to atom j and $r_{ij} = |\mathbf{r}_{ij}|$, ie,

$$r_{ij} = \sum_{k=1}^{3} \sqrt{(x_{i_k} - x_{j_k})^2}. \tag{1.6}$$

The total force on atom i is the sum of forces from all other atoms in the system as given by

$$\mathbf{f}_i = \sum_{j \neq i} \mathbf{f}_{ij}. \tag{1.7}$$

To perform MD simulations, a system of equations of motion must be solved numerically. Here a variety of different techniques may be employed, such as the Verlet algorithm or predictor-corrector methods. In this study, MD code is implemented in MATLAB using the built-in ode45 differential equation routine to numerically solve the governing equations. Further details are provided in subsequent chapters.

1.4 Lattice Defects

As the study of the mechanical properties of graphene has become important in materials science, so has the study of defects within a graphene lattice. A lattice defect is any imperfection in a lattice of atoms. Lattice defects exist in many forms and are generally classified into three different types: incomplete bonding defects, topological defects, and heterogeneous defects. Incomplete bonding defects include vacancies, dislocations, or other defects in which atoms are removed from systems, resulting in broken bonds between atoms. Topological defects refers to irregular lattice geometry introduced to the system, and heterogeneous defects are those that result from doping, or introducing different atom types to a lattice [17]. The study of these defects has become increasingly important in nanotechnology since the presence of defects in a lattice can have a strong influence upon the mechanical behavior of the system. Because of this, both analytical and numerical studies have been conducted that aim to understand this influence.

Three types of incomplete bonding defects will be considered in this thesis: vacancies, dislocations, and interstitials. A vacancy is a lattice defect in which a single atom is removed from an equilibrated lattice. There may also be multi-vacancies in which multiple atoms are removed from the lattice. In a graphene sheet, this defect results in several broken covalent bonds near the vacancy. In a system governed by the Lennard-Jones potential, however, this defect does not break bonds because it is a non-bonded potential. The second defect type, a dislocation, can be constructed, for example, by removing about half of the atoms in a single row, and then closing the resulting gap in such a way so that the lattice is almost perfect everywhere except for the tip of the gap. The final type, an interstitial, is a defect in which an extra atom is inserted into a region between the atoms of a perfect lattice.

Each of these defects causes a deformation of the lattice that can only be removed by removing the defect, for example, via diffusion. In graphene, these defects have been shown to significantly impact not only its mechanical, but also other properties. For example, graphene defects have been shown to result in degraded particle transport [18], giving rise to magnetic moments [1], and result in lattice deformation [19]. Further studies of defects are explored in Chapter 4. Figure 1.4 depicts each of the defects in a graphene sheet. Vacancy and dislocation defects have also been observed in graphene via high resolution transmission electron microscopy (TEM), [2], evidence of which is provided in Figure 1.5.

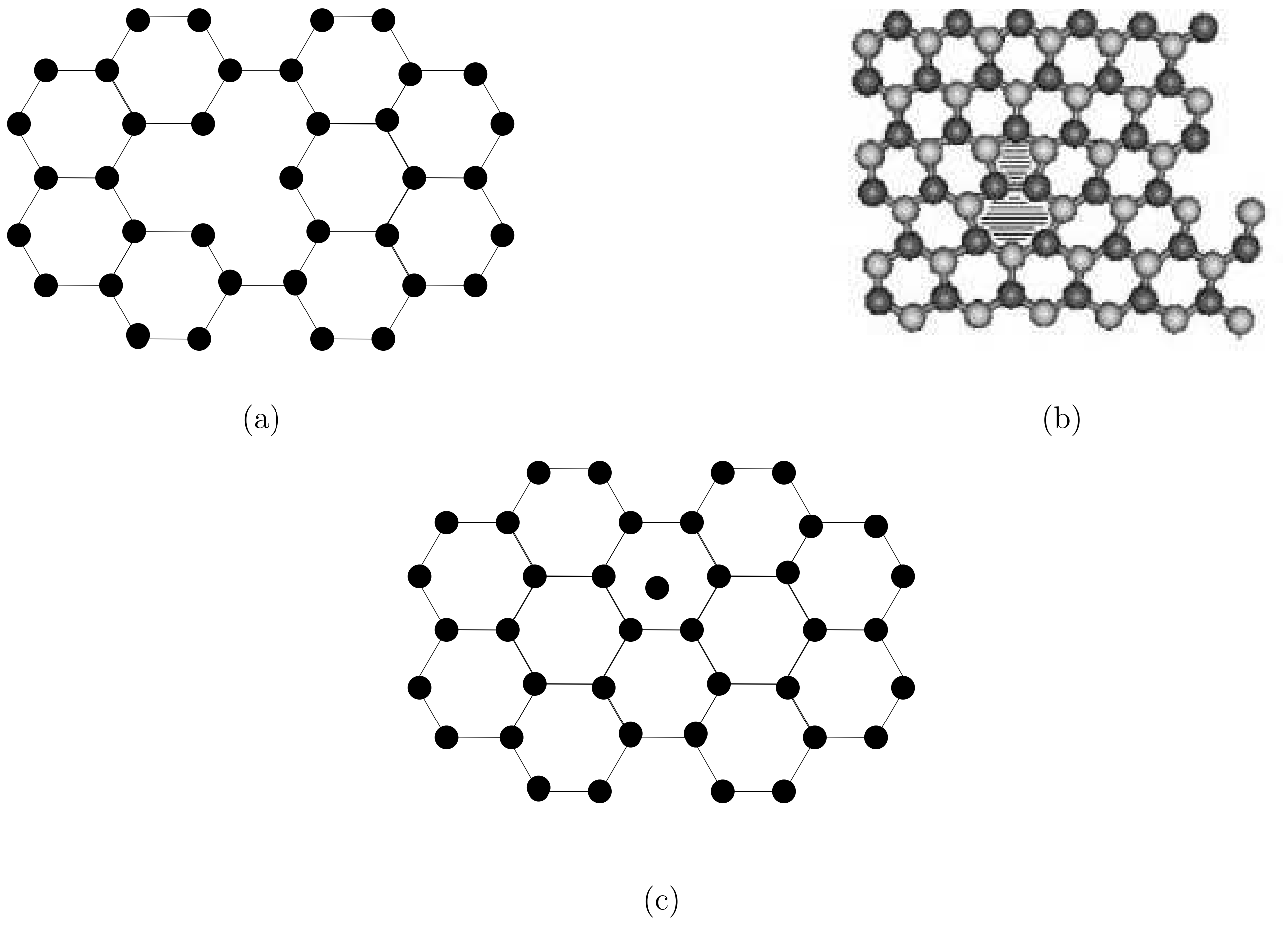
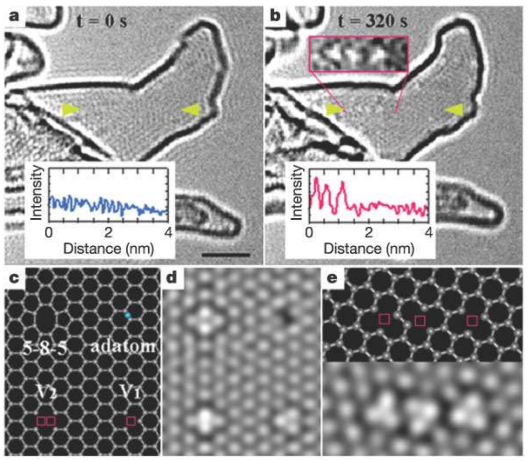
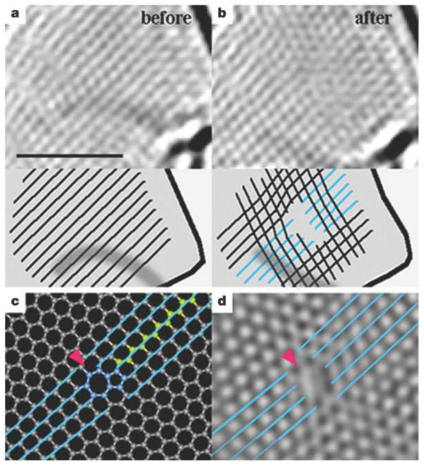


Figure 1.4: Graphene sheet with a (a) vacancy defect (b) dislocation defect, (Carpio [1]), and (c) interstitial defect

These defects will be studied in a lattice governed by the Lennard-Jones potential, with the ultimate goal to model defects in graphene. The Lennard-Jones potential is simple to incorporate in MD simulations, so it is used here. In future work the Lennard-Jones potential could be replaced with a REBO potential, which models interactions within a carbon lattice.



(a)



(b)

Figure 1.5: High-resolution TEM and simulated TEM images of graphene layers with (a) vacancies and (b) dislocation, (Hashimoto, [2])

1.5 Summary of Thesis

The purpose of this thesis is to use MD simulations to model a lattice governed by the Lennard-Jones potential. Defects of the lattice structure— vacancies, dislocations, and interstitials— are simulated in order to understand the evolution of an imperfect lattice over time. As an introductory step, in Chapter 2, one-dimensional MD simulations are implemented. This one-dimensional model describes a straight chain of n atoms. Three different boundary conditions are implemented in this setting:

1) stationary atoms at both ends of the chain,

2) fixed walls surrounding both sides of the chain, and

3) periodic boundaries.

Each of these boundary conditions result in a different behavior, as evidenced by the final positions of atoms in a chain. In Chapter 3, the model is expanded to perform two-dimensional MD simulations. In two dimensions, the chain of atoms is replaced by a lattice of atoms. An equilibrated Lennard-Jones lattice is constructed, and the behavior of the lattice under various perturbations is explored. Then, the three boundary conditions are implemented in two dimensions and the resulting configurations of atoms are investigated.

In Chapter 4, lattice defects are modeled in two dimensions by using the appropriate perturbations of an equilibrated Lennard-Jones lattice as the initial conditions. The dynamics of defective Lennard-Jones lattices will be observed by tracking the positions of atoms over time. Finally, in Chapter 5 the reactive empirical bond order (REBO) potential is discussed. The REBO potential can be used to model covalently bonded carbon bonds to model interactions between atoms within a graphene sheet, with the ultimate goal of understanding the role of lattice defects in a graphene layer. We conclude with some suggestions for further research.

CHAPTER II

ONE-DIMENSIONAL SIMULATIONS

As an introduction to molecular dynamics, we describe MD simulations of a one-dimensional chain of atoms. We develop the governing numerical equations and consider three types of boundary conditions: stationary atoms on both sides of the chain, fixed walls surrounding both sides of the chain, and periodic boundaries. Simulations are conducted for each boundary condition using the same initial conditions and parameters, so that the difference in behavior can be observed. All code for this one-dimensional study is written and executed in MATLAB, and also provided in Appendix A.

2.1 Model of a Chain of Atoms

To construct a one-dimensional system, a chain of n atoms is considered, as shown in Figure 2.1. The value of n is an input parameter so that the number of atoms, and thus the length of the chain, can be varied. Each atom in the chain is subject to pairwise LJ-type interaction with every other atom in the chain. It is also assumed that, due to their LJ-type interactions, atoms cannot exchange order within the chain.

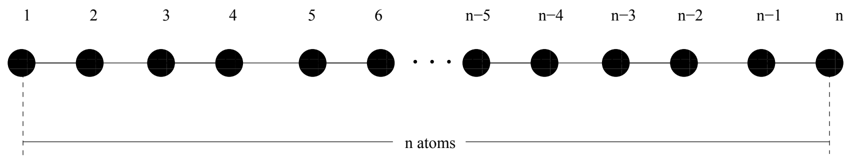


Figure 2.1: Diagram of a one-dimensional chain of atoms

We can think of each pair of atoms as connected by a spring. The motion of each individual atom is governed by Newton's second law of motion,

$$f = ma, \tag{2.1}$$

where f is force, m is mass, and a is acceleration. If the position of an atom at time t is given by $x(t)$, then

$$a = \ddot{x} \tag{2.2}$$

and

$$f = -U_x, \tag{2.3}$$

so that the equation of motion of the atoms is

$$m\ddot{x} = -U_x. \tag{2.4}$$

Since we are principally interested in equilibrium configurations, a term that describes damping is also introduced. The simplest way to model damping is to assume that it is proportional to the velocity. So, the equation of motion becomes

$$m\ddot{x} + \beta\dot{x} = -U_x, \tag{2.5}$$

where β is a positive damping coefficient. Equation (2.5) is the differential equation used in conventional MD simulations, where U is the function describing the interatomic potential. As explained previously, in this this study, the Lennard-Jones potential is used to represent the pairwise interaction between atoms,

$$U(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right], \tag{2.6}$$

where

$$r = |x - y| \tag{2.7}$$

is the distance between two atoms positioned at x and y, respectively.

Given the equation of motion (2.5) and the Lennard-Jones potential (2.6), the governing equation for the atom with position x is

$$m\ddot{x} + \beta\dot{x} = -24\epsilon\left[2\frac{\sigma^{12}}{r^{13}} - \frac{\sigma^{6}}{r^{7}}\right]. \tag{2.8}$$

Based on this second order equation, a first order system of equations describing a chain of n atoms can be constructed.

2.2 One-Dimensional Numerical Model

To model the motion of a system of atoms within an MD simulation, we construct and discretize the system of governing ODEs based on equation (2.8). We rewrite equation (2.8) as

$$m\ddot{x}_i = -f_i - \beta\dot{x}_i, \tag{2.9}$$

where i denotes the i^{th} atom. The force acting upon each individual atom, i, in the system, is the sum of forces resulting from its pairwise interactions with all other atoms, $j \neq i$, in the system, so that

$$f_i = \sum_{j=1; j \neq i}^{n} U_{x_i}(|x_i - x_j|). \tag{2.10}$$

The force with which an atom j acts on the atom i is given by the Lennard-Jones force in (2.11) so that

$$U_{x_i}(|x_i - x_j|) = -24\epsilon \left[2\frac{\sigma^{12}}{(|x_i - x_j|)^{13}} - \frac{\sigma^6}{(|x_i - x_j|)^7} \right]. \tag{2.11}$$

Because the differential equation (2.9) is of second order, it can be written as a system of two first order differential equations. Indeed, set

$$\dot{x} = v_i \tag{2.12}$$

so that the first-order system becomes

$$\frac{\partial x_i}{\partial t} = v_i \tag{2.13}$$

$$\frac{\partial v_i}{\partial t} = -\frac{1}{m} f_i - \frac{\beta}{m} v_i. \tag{2.14}$$

where $i = 1, ..., n$. This system of $2n$ first-order equations can be solved by executing the MATLAB ode45 solver. The simulation is set up so that the chain of atoms may be initialized by defining the initial positions and velocities of each atom in the system. The system parameters, including m, σ, and ϵ, can be initialized according to the type of atoms composing the system. The value of the damping coefficient, β, can also be varied. In this thesis we assume that $0 \leq \beta \leq 1$ $eV(ps/\AA)$, where $\beta = 1$

represents maximum damping, so that the equation becomes

$$m\ddot{x}_i = -f_i - \dot{x}_i. \tag{2.15}$$

The minimum value of $\beta = 0$, corresponds to undamped motion. For $\beta = 0$, the governing equation becomes

$$m\ddot{x}_i = -f_i. \tag{2.16}$$

Parameters and system definitions are initialized in the main file, OneD_MD-System.m. Following these initializations, the initial conditions are prescribed, including the positions and velocities of all atoms in the chain. Next ode45 is called, using the function that defines the numerical system of equations— this is the most computationally expensive step. For the case with no boundary conditions, the function file is LJODDE_noBC.m. This function defines the right-hand side of the system of ODEs and calculates the force acting upon each atom by defining the force calculations according to Lennard-Jones potential, given by (2.10), so that the force acting upon each atom is the sum of forces resulting from pairwise interactions with every other atom in the system. To calculate the LJ force between two atoms given the distance between the two atoms, the ForceLJ.m file is called. Given the total force acting upon each atom, its positions and velocities at the following time step are then calculated. This process is repeated for a discrete number of time steps until the preset final runtime is reached.

Using this code, several test simulations are executed in order to observe the behavior of a chain of atoms. Based on the Lennard-Jones force, it is expected that, if the spacing between two atoms is less than r_{eq}, the atoms will repel; if the spacing between them is greater than r_{eq}, the atoms will attract. To observe this behavior, a system of $n = 21$ atoms is constructed with initial positions such that some pairwise interactions have spacing less then r_{eq}, while others have spacing greater than r_{eq}. Each atom is initialized with zero velocity. The initial positions and the initial separation between adjacent atoms are provided in Table 2.2. Also listed in Table 2.1 are the values for other parameters used in the MD simulation. Later, the same $n = 21$ chain of atoms with the same initial conditions and parameters is used to observe the behavior of atoms under the various boundary conditions.

Table 2.1: Variables and parameters used in the one-dimensional numerical model

Variable/Parameter	Description	Value
n	Number of atoms	21 atoms
m	Atomic mass	1.0 amu
σ	Separation at zero potential	1.0 Å
ϵ	Well depth (strength of interaction)	1.0 kJ/mol
β	Damping coefficient (strength of damping)	1.0 $eV(ps/\AA)$

As seen in Table 2.2, the distances separating atoms 1 and 2, and $n-1$ and n are significantly less than r_{eq}— approximately $.632449 r_{eq}$ and $.827912 r_{eq}$, respectively. Since these distances are less than r_{eq}, it is expected that the two end atoms, 1 and n, will be repelled from the remaining atoms, expanding the length of the chain. This repulsion is evident in Figure 2.2, which illustrates the positions of the atoms over

Table 2.2: Initial positions and separation between atoms in the test case

Atom, i	x_i	Distance, $x_{i+1} - x_i$	Atom, i	x_i	Distance, $x_{i+1} - x_i$
1	0.4908	0.7099 Å	11	11.3377	1.2017 Å
2	1.2007	1.4720 Å	12	12.5394	1.2216 Å
3	2.6727	1.0171 Å	13	13.7610	0.9569 Å
4	3.6898	0.9882 Å	14	14.7179	1.1418 Å
5	4.6780	1.0298 Å	15	15.8597	1.2858 Å
6	5.7078	1.2411 Å	16	17.1455	0.9465 Å
7	6.9489	1.1493 Å	17	18.0920	1.4020 Å
8	8.0982	0.9418 Å	18	19.4940	1.2016 Å
9	9.0400	1.3569 Å	19	20.6956	0.9963 Å
10	10.3969	0.9408 Å	20	21.6919	0.9293 Å
--	--	--	21	22.6212	---

the simulation time. The slight attraction between adjacent atoms that are initially separated by distances larger than r_{eq} can also be observed in this figure.

The simulation of the free chain with no boundaries shows that when atoms are initially separated by distances less than r_{eq}, the atoms at the end of the chain can be expelled from the rest of the chain, so boundary conditions are necessary in order to confine atoms to a defined interval. Thus, the three boundary conditions— stationary atoms on both sides of the chain, walls surrounding both sides of the chain, and periodic boundaries are investigated. It is also found that a long runtime is required to complete the MD simulation. The runtime increases with larger systems in higher dimensions, so neighbor list construction is used to neglect negligible interactions. We

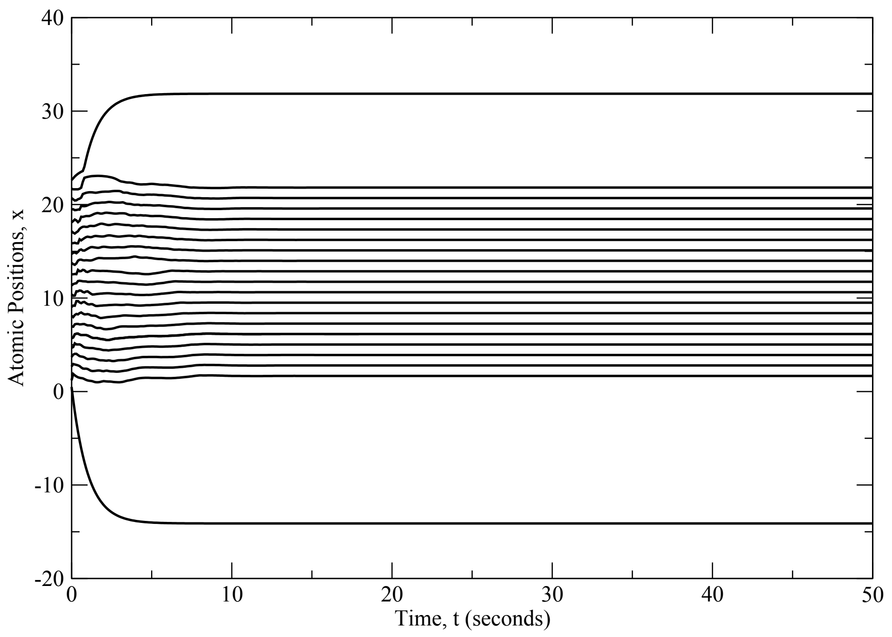


Figure 2.2: Position vs time graph of a system of $n = 21$ atoms with no boundary conditions

discuss neighbor list construction next, followed by the implementation of boundary conditions.

2.3 Neighbor List Construction

MD simulations can take a long time to complete, since there is a large number of operations conducted at each individual time step. The number of operations is dependent upon the size of the system, so as the size of the system increases, so does the number of calculations. Each atom in the system interacts with every other atom and their corresponding forces must be calculated at every time step. Thus, for a system of n atoms,

$$n(n-1) = n^2 - n \tag{2.17}$$

interactions at every time step are required. As n increases, the number of these operations increases. To remove unimportant calculations from the simulation in order to improve runtime, neighbor list construction is used.

Neighbor list construction is a common method for numerical approximation used in MD to decrease computational expense and increase efficiency. This is done by constructing lists of neighboring atoms that share nonzero LJ-type interactions, while ignoring pairwise interactions that have zero, or nearly zero, force. Under this assumption, the Lennard-Jones force can be rewritten to account for the cutoff as

$$F(r) = \begin{cases} -24\epsilon\left[2\frac{\sigma^{12}}{r^{13}} - \frac{\sigma^6}{r^7}\right], & r \leq r_c \\ 0, & r > r_c, \end{cases} \tag{2.18}$$

where r_c is some prescribed cutoff distance. As referenced in earlier discussion, as atoms move further away from each other, the Lennard-Jones potential decays rapidly to zero. By defining the force in such a way, the potential function no longer gradually decays to zero, but suddenly equals zero at the cutoff. This allows us to approximate the solution by ignoring negligible pairwise interactions between atoms separated by distances large enough so that the force between them is approximately zero. A cutoff distance, r_c, is also defined to identify which pairwise interactions between two atoms are accounted for, and which are ignored. If two atoms are separated by less than r_c, than the indices of the atoms are stored in the neighbor list. If the distance between two atoms is greater than the cutoff distance, the pairwise interaction is ignored.

To include the neighbor list construction in the simulations, a new MATLAB function is written— neighborList.m— which, given the positions of atoms and the prescribed cutoff radius, creates a neighbor list of all the pairwise interactions. This is achieved by calculating the distances between every i^{th} and j^{th} atom. If the distance between the two atoms is less than the cutoff distance, then the indices of each atom are stored in the neighbor list to account for their pairwise LJ-type interaction. Otherwise, the interaction is treated as negligible and omitted. The neighbor list is then constructed as a two-column vector in which each row contains indices of atoms for which the force resulting from their pairwise interaction is calculated. Force calculations can now be made based on the indices stored in the neighbor list. For k rows in the neighbor list vector, there are k force calculations, rather than looping through $n(n-1)$ force calculations.

In implementing neighbor list construction, there are two important considerations. The first is the value of the cutoff distance, r_c. If the cutoff is too small, then significant interactions may be left out, leading to inaccurate results. If the cutoff is too large, then zero forces may be calculated, leading to unnecessary runtime expense. The second is the number of neighbor list reconstructions to calculate throughout the runtime. This is important to consider because, as runtime progresses, atoms may move further apart from each other, allowing the neighbor list to become smaller. Alternatively, atoms may move closer together and without reconstructing the neighbor list, important interactions in the simulation can be missed, giving inaccurate results. We also note that many calculations are executed to construct a neighbor list, so increasing the number of reconstructions also increases the overall runtime. By prescribing the number of times the neighbor list should be reconstructed throughout a simulation, errors can be avoided while optimizing runtime.

To determine an optimal value for the cutoff, we run simulations with cutoff distances $r_c = 6$, $r_c = 5$, and $r_c = 4$ Å to find the resulting approximation error and runtime. For $r_c = 6$ Å, the Lennard-Jones force is $f(6) = .000086$, while for $r_c = 5$ Å, $f(5) = .000307$, and for $r_c = 4$ Å, $f(4) = .001464$. To test the accuracy resulting from each cutoff value, simulations are run using a system of $n = 21$ atoms with uniform separation $.8r_{eq}$. The resulting final positions of atoms for each cutoff value are calculated and compared to the results from a simulation without neighbor list construction. Also, a simulation with $r_c = 20 \times (.8r_{eq})$ Å, which is the total length of the chain, is run to ensure the accuracy of neighbor list construction, since the results should be identical to the simulation without neighbor list construction. In Table 2.4 the final positions for each simulation are given, and the percent differences in comparison to the simulation without neighbor list construction are provided. Table 2.4 appears at the end of this section.

To test the efficiency of the neighbor list construction, simulations are executed with neighbor list construction for each cutoff value. The total runtime in seconds for each case is given in Table 2.3. A percent difference is provided as well to demonstrate the significant decrease in runtime. For these simulations, the systems are initialized so that the spacing between each atom is uniform throughout the chain. The systems have $n = 200$ atoms, with $\sigma = 1$, $\epsilon = 1$, $m = 1$, and $\beta = 1$. The large percent differences in runtime show that the simulations utilizing neighbor list construction run much faster. Furthermore, it is shown that the runtime decreases for smaller cutoff distances.

Table 2.3: Runtime (in seconds) comparison for simulations with $r_c = 6$, $r_c = 5$, and $r_c = 4$ Å.

	No NL	$r_c = 6$ Å		$r_c = 5$ Å		$r_c = 4$ Å	
Spacing	Runtime	Runtime	% Diff	Runtime	% Diff	Runtime	% Diff
$.8r_{eq}$ Å	538.5624	34.2433	93.6417%	26.2645	95.1232%	21.8122	95.9499%
$.95r_{eq}\AA$	439.3861	26.9580	93.8646%	22.2092	94.9454%	17.1823	96.0895%
r_{eq} Å	363.6068	23.3639	93.5744%	17.9354	95.0674%	14.2436	96.0827%
$1.05r_{eq}\AA$	427.4845	26.3390	93.8386%	20.4484	95.2166%	15.9695	96.2643%
$1.2r_{eq}\AA$	1054.5653	51.2365	95.1415%	37.1851	96.4739%	27.5255	97.3899%

Given the accuracy of results for various cutoffs in Table 2.4 and the improvement of runtime efficiency for the same cutoffs in Table 2.3, we choose the cutoff $r_c = 5$ Å for our simulations. Clearly, as shown in Table 2.3, using neighbor list construction significantly reduces runtime. However, as shown in Figure 2.4, a cutoff of $r_c = 4$ Å results in some errors above 1%, and even as high as 5%. On the other hand, $r_c = 6$ Å and $r_c = 5$ Å provide reasonable approximations, as all resulting errors are less than 0.75%. However, the cutoff $r_c = 5$ Å also provides a faster runtime than $r_c = 6$ Å. To optimize the value of cutoff distance, we choose a distance that results in small error, but also increased runtime. For this reason, we choose $r_c = 5$ Å, since it provides reasonable approximation with significantly improved runtime efficiency.

Table 2.4: Final positions comparison table for $r_c = 20 \times (.8r_{eq})$, $r_c = 6$, $r_c = 5$, and $r_c = 4$ Å.

	No NL	$r_c = 20(.8r_{eq})$ Å		$r_c = 6$ Å		$r_c = 5$ Å		$r_c = 4$ Å	
Atom, i	x_i	x_i	% Diff	x_i	% Diff	x_i	% Diff	x_i	% Diff
1	-4.9234	-4.9234	0%	-4.9259	.0508%	-4.9267	.0670%	-4.9307	.1483%
2	-3.8025	-3.8025	0%	-3.8050	.0657%	-3.8057	.0841%	-3.8098	.1920%
3	-2.6830	-2.6830	0%	-2.6856	.0969%	-2.6863	.1229%	-2.6903	.2721%
4	-1.5637	-1.5637	0%	-1.5662	.1599%	-1.5669	.2046%	-1.5709	.4604%
5	-0.4443	-0.4443	0%	$-.4469$	.5852%	-0.4476	.7427%	-0.4516	1.6430%
6	0.6751	0.6751	0%	.6726	.3703%	0.6719	.4740%	0.6679	1.0665%
7	1.7961	1.7961	0%	1.7935	.1448%	1.7928	.1837%	1.7889	.4009%
8	5.6199	5.6199	0%	5.6199	0%	5.6199	0%	5.6198	.0018%
9	6.7409	6.7409	0%	6.7409	0%	6.7409	0%	6.7408	.0015%
10	7.8603	7.8603	0%	7.8603	0%	7.8603	0%	7.8603	0%
11	8.9797	8.9797	0%	8.9797	0%	8.9797	0%	8.9797	0%
12	10.0991	10.0991	0%	10.0991	0%	10.0991	0%	10.0991	0%
13	11.2185	11.2185	0%	11.2185	0%	11.2185	0%	11.2186	.0009%
14	12.3395	12.3395	0%	12.3395	0%	12.3395	0%	12.3395	0%
15	16.1633	16.1633	0%	16.1659	.0161%	16.1666	.0204%	16.1705	.0445%
16	17.2843	17.2843	0%	17.2868	.0145%	17.2875	.0185%	17.2915	.0417%
17	18.4037	18.4037	0%	18.4063	.0141%	18.4070	.0179%	18.4110	.0397%
18	19.5231	19.5231	0%	19.5256	.0128%	19.5263	.0164%	19.5303	.0369%
19	20.6424	20.6424	0%	20.6450	.0126%	20.6457	.0160%	20.6497	.0354%
20	21.7619	21.7619	0%	21.7644	.0115%	21.7651	.0147%	21.7692	.0335%
21	22.8828	22.8828	0%	22.8853	.0109%	22.8861	.0144%	22.8901	.0319%

2.4 Boundary Conditions: Stationary Atoms at the Ends of a Chain

The first boundary condition that is implemented is a case in which positions x_1 and x_n of the two end atoms (atoms 1 and n) are held fixed. The interior atoms, 2 through $n-1$, interact between themselves and with the ends atoms, and move freely inside the interval (x_1, x_n). Since atoms will be confined to a prescribed finite domain, the distance between two atoms cannot exceed the length of the chain. Figure 2.3 provides a schematic of this configuration.

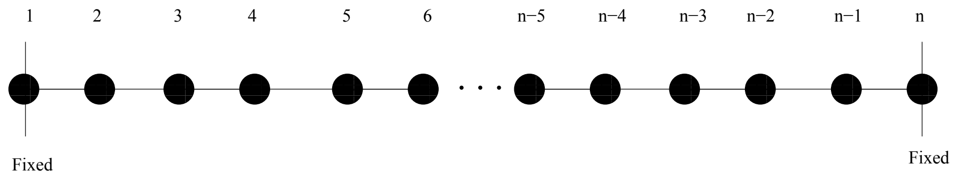


Figure 2.3: Schematic of the chain of atoms with stationary end atoms

In the previous simulations with no boundaries, the system of two equations is solved per atom, totaling $2n$ equations at every time step. For this case, the system is now solved for $n-2$ atoms since we now exclude the end atoms. This totals $2(n-2)$ equations that are solved at every time step. We assume that there is zero net force acting upon atoms 1 and n. Also to ensure that the two end atoms remain fixed throughout the simulation, their initial velocities must be set to zero. With zero initial velocity and zero force acting upon them, the end atoms will remain in their initial positions for the entire runtime.

As in the previous studies, all initializations are executed in the main functions OneD_MDSystem.m. The ode function used here is LJODE_fixed.m, which varies slightly to LJODE_NoBC.m, principally because the system is now solved for atoms 2 through $n-1$ rather than atoms 1 through n.

Starting from the same initial positions and velocities, as in the simulation without boundary conditions, the behavior of the chain can be observed under the fixed-end boundary. In the simulation without boundary conditions, the distribution of initial positions lead to large repulsive forces upon the end atoms. This is a consequence of our assumption that the initial separation distances between the end atoms and their immediate neighbors are much smaller than r_{eq}. With the positions of end atoms fixed, they can no longer be repelled away from the chain. The rest of the atoms move freely between the fixed end atoms, oscillating briefly until the system equilibrates due to dissipation. The position vs. time graph of this simulation is given in Figure 2.4.

Note that the final positions of atoms in this case depend on the positions of end atoms, 1 and n. Thus, the resulting equilibrium configuration is dependent upon the compression imposed by the physical boundaries. To demonstrate this idea, the same system as above is simulated, however the end atoms are now given different positions. First, we simulate the system in which the distances separating atoms 1 and 2, and n and $n-1$ are decreased from the previous simulation. See Figure 2.5. As observed, there is a lot more oscillation before the atoms eventually equilibrate. Their equilibrium configuration is different in that there is less distance separating

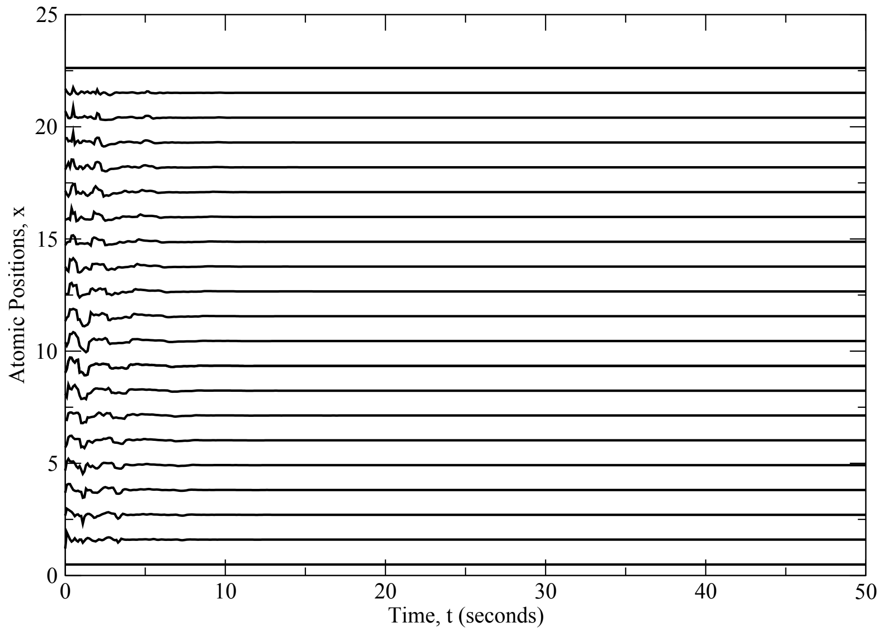
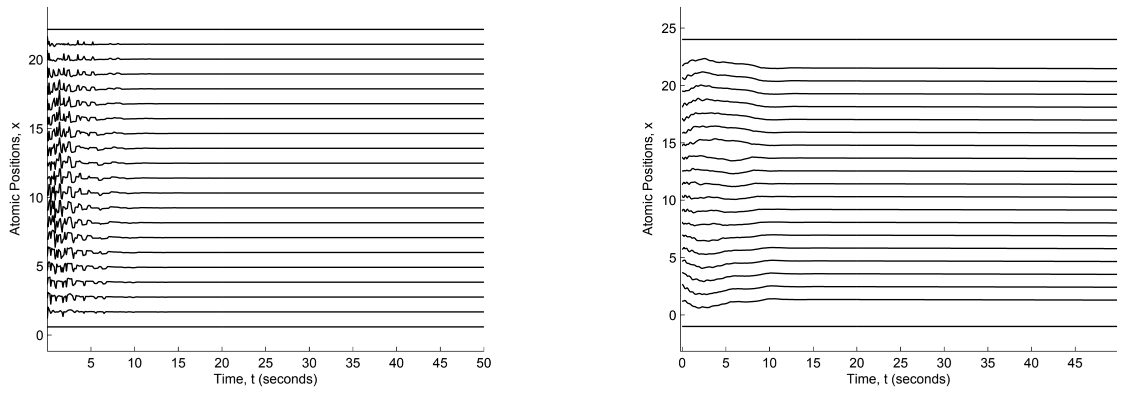


Figure 2.4: Position vs time graph of a system of $n = 21$ atoms with stationary end atoms

atoms at their final positions. In the second simulation, the distances between atoms 1 and 2, and n and $n - 1$ are now increased from the first simulation for this boundary condition. This time, there is less oscillation between atoms. Their equilibrium configuration also clearly results in non uniform spacing between atoms. The graphs for each simulation are provided in Figure 2.5.

The results illustrated by Figure 2.5 show that the equilibrium configuration for fixed boundaries depends on the length of the chain. For a larger length, there are larger equilibrium distances, breaking the chain into separate parts. For smaller lengths there are smaller equilibrium distances. Thus, it is feasible to have equilibrium configurations with non uniform spacing between atoms. To remove this dependence,



(a) (b)

Figure 2.5: Position vs time graphs of a system of $n = 21$ atoms with (a) a compressed chain, and (b) an elongated chain

an in effort to simulate systems with uniform spacing, we investigate further boundary conditions.

2.5 Boundary Conditions: A Chain Between Two Walls

For this type of boundary conditions, 'walls' are placed on either side of the chain of atoms so that the atoms are able to move freely between the walls, but cannot penetrate them (Figure 2.6). The wall positions can be varied between different simulations, but are fixed during a given simulation. Note that this setup essentially differs from that in the previous section in the type of interaction imposed on the boundary links of the chain. When the atoms at the ends of the chain are fixed, that interaction is of Lennard-Jones type, while here the interaction only occurs during a collision of an end atom with a wall. The velocity of an atom after collision with a wall depends on whether the wall is hard or soft, as we explain in the following discussion.

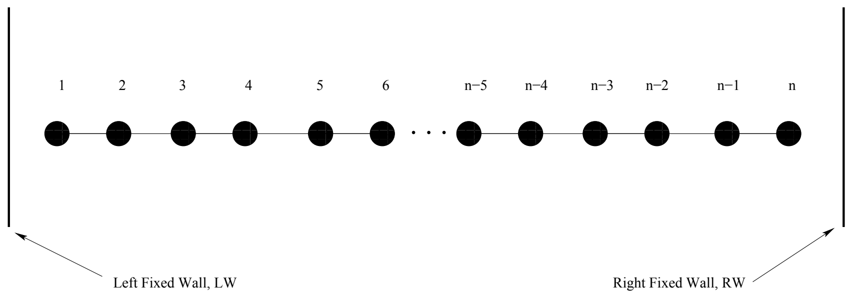


Figure 2.6: Schematic of the chain of atoms with walls boundary condition

In addition to the same initializations as in the previous cases, the positions of the walls must be specified so that

$$x_{LW} < x_1, \tag{2.19}$$

where x_{LW} is the position of the left wall, and x_1 is the position of the first atom, which is adjacent to the left wall. Also, x_{RW} is the position of the right wall so that

$$x_{RW} > x_n, \tag{2.20}$$

in which x_n is the position of the final atom in the chain, which is adjacent to the right wall. We introduce a parameter, k, to represent the hardness of both walls. This parameter varies between 0 and 1. When a wall is "soft", $k = 0$ and an atom colliding with the wall will stick with zero velocity until pulled off by attractive forces from other atoms in the chain. In this case, the colliding atom loses all its kinetic energy upon impact with the wall. If $k = 1$, then the wall is "hard" and an atom colliding with the wall with the velocity v will bounce off the wall in the opposite direction with velocity $-v$. Here, the kinetic energy of the colliding atom is preserved. For intermediate k values between 0 and 1, a colliding atom with velocity v will bounce off the wall with velocity $-kv$. This is achieved by letting v_f be the velocity after hitting the wall so that

$$v_f = -kv_0, \tag{2.21}$$

where v_0 is the velocity of the colliding atom before it hits the wall.

The main difficulty in implementing this boundary condition is determining the instances of wall collisions throughout the simulation. This is done by utilizing a MATLAB events function, which identifies times when an atom collides with a wall. If a collision is detected by the nested events function, wallevents, the ode45 solver stops and returns to the main function, OneD_MDSystem.m to subsequently execute four tasks: reposition the colliding atom, calculate the rebound velocity of the colliding atom, calculate the nearest concurrent discrete time step, and append the time and final solution vectors. The atom is first repositioned, given its position x_i, by

$$x_i = x_{LW} + 10^{-12}\ \mathring{A} \tag{2.22}$$

if it hits the left wall, and

$$x_i = x_{RW} - 10^{-12}\ \mathring{A} \tag{2.23}$$

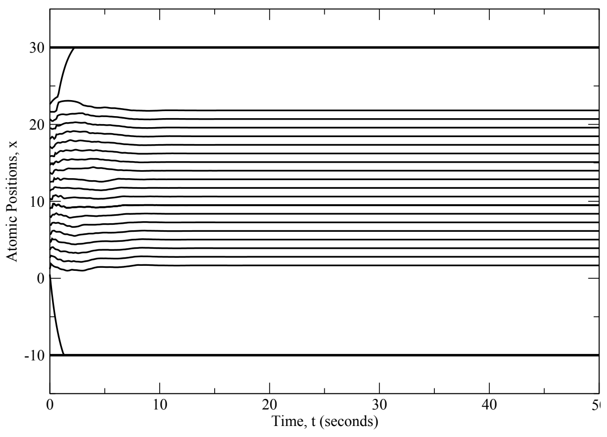
if it hits the right wall. This is done so that the atom is not placed directly on the wall, but very close to it. The reason for this step is computational—if an atom remains directly on the wall, the events function will be triggered repeatedly. Thus, we approximate the position of the colliding atom very close to the wall so that the events function is no longer triggered. After the atom is repositioned, its velocity is then set based on the wall hardness, as given in (2.21). The next discrete time step must now be calculated, so that it can be passed back into ode45 as the new starting point to continue the remaining simulation with the originally prescribed time vector. This is achieved by calling the function timestep.m, which is written to return the next discrete time step and the number of remaining time steps, given the time in which the atom hit the wall. Finally, the current time and current solution vectors are appended to the time and solution vectors. This is done because the number of times an atom will contact a wall is unknown a priori, so the final position and time vectors must be appended every time ode45 stops. After these four tasks are completed, ode45 is called again, passing in the newly calculated positions, velocities, and time, along with the other necessary parameters. This process repeats for any number of wall events until the end of the simulation.

For wall boundaries, the same main function, OneD_MDSystem.m is used and the ode function passed into ode45 is LJODE_walls.m, which defines the calculations of the forces for pairwise interactions under this boundary condition. As explained, the MATLAB events function is also employed in the code in order to identify when an atom hits a wall. To use the events function, a new nested function, wallevents, is created, which identifies when an atom hits a wall, given the vector of positions of atoms. This function must be passed into ode45 as an options parameter, so that the positions of atoms are checked at every time step.

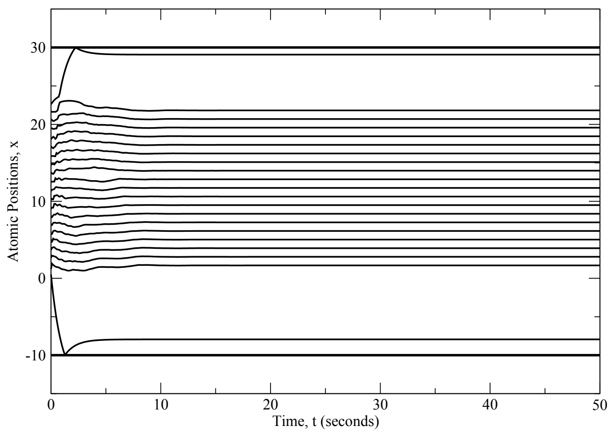
Using the same system of atoms as in the previous sections, the simulation with the walls boundary condition is executed in order to understand the effects of the walls on the evolution of the chain of atoms. The initial conditions are provided in Table 2.2, and walls are positioned at $x_{LW} = -10$ Å and $x_{RW} = 30$ Å. The code is executed with $k = 0$, $k = 0.5$, and $k = 1$. The results of each simulation are provided in Figure 2.7. In each case it is expected that the two end atoms hit their adjacent walls due to initial repulsion, but the rebound resulting from the collision depends on k.

As illustrated in the Figure 2.7, the chain remains confined to the interval between the walls and all atoms move freely between the fixed walls. For $k = 0$, the atoms stick to the walls and remain there because they have zero velocity. For $k = 0.5$ and $k = 1$, the rebound of the atoms off the walls show that larger k values result in a larger rebound off the walls. In every case, the system eventually equilibrates due to dissipation of the velocities of atoms.

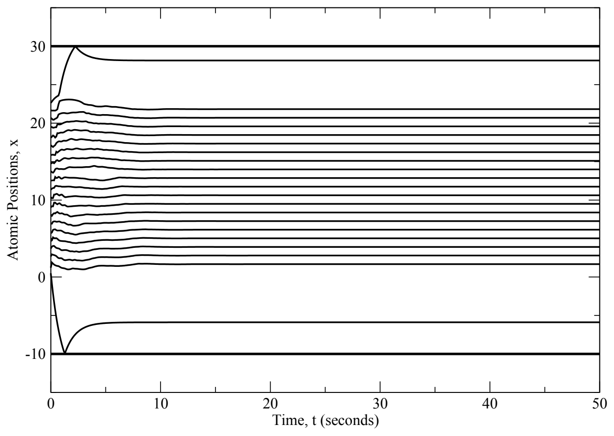
We also note that as in the previous boundary condition, the resulting equilibrium configuration with wall boundaries depends on the positions of the walls. To demonstrate this, two simulations are ran for the same system of $n = 21$ atoms with $k = 1$, but with different wall positions. In the first simulation we let $x_{LW} = 0$ Å and $x_{RW} = 25$ Å so that there is less distance separating the two walls, confining the chain



(a)



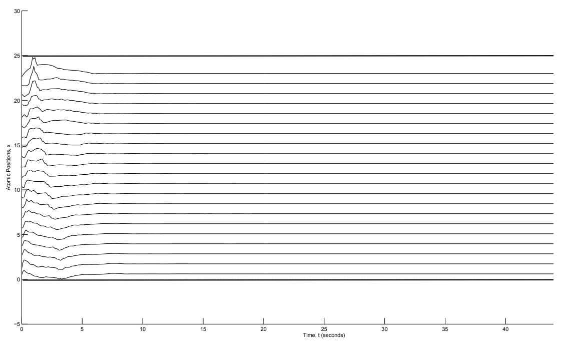
(b)

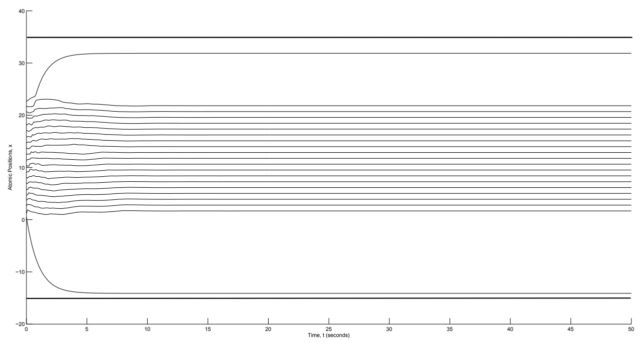


(c)

Figure 2.7: Position vs time graphs of a system of $n = 21$ atoms with walls, where (a) $k = 0$, (b) $k = 0.5$, and (c) $k = 1$

to a smaller space. Here we see that the chain equilibrates with different positions of atoms than in the previous simulation. Next, we place the walls at $x_{LW} = -15$ Å and $x_{RW} = 35$ Å in order to increase the space in which atoms can move. Here, the equilibrium configuration is the same as in the no boundaries case, since the walls are sufficiently far apart so that the atoms do not interact with them. Graphs for each of the simulations are provided in Figure 2.8.





(a) (b)

Figure 2.8: Position vs time graphs of a system of $n = 21$ atoms with (a) decreased wall separation and (b) increased wall separation

As seen in Figure 2.8, each simulation results in different equilibrium configurations. With wall boundaries, the equilibrium configuration then depends on the positions of the walls, x_{LW} and x_{RW}. Furthermore, the final positions of atoms depend on the value for k. Thus, in this case, the equilibrium configuration depends on both the placement and hardness of physical boundaries. Wall boundaries are useful in cases where the focus is upon the interactions of atoms with the boundaries. In a typical MD simulation of a 1000 atoms or more, a very small percentage of the atoms directly experience the boundary, although the boundary condition can have a profound effect on an equilibrium configuration of atoms. Further, in an equilibrium configuration, the forces experienced by atoms on the boundary and in the interior of the chain are different, which leads, for example, to different equilibrium interatomic distances in the interior and edges of the chain. To remove this effect, one can impose periodic boundary conditions, which we consider in the next section.

2.6 Periodic Boundary Conditions

The final set of boundary conditions explores a system that is free of physical boundaries, such as free boundaries, walls, or stationary end atoms, but is still confined to a bounded space. This is accomplished by implementing periodic boundary conditions representing an infinite number of identical copies of the system, in both the positive and negative directions. This setup essentially corresponds to a chain of atoms wrapped around a cylinder. In this case the end atoms, 1 and n, describe the same atom and there are infinitely many interactions that occur with infinitely many copies of the chain. Figure 2.9 shows the chain with periodic boundary conditions.

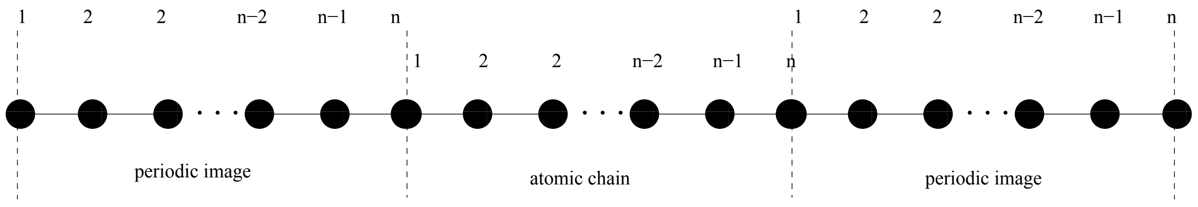


Figure 2.9: Schematic of a chain of atoms with periodic boundary conditions

To account for all atomic interactions, we classify three different scenarios that occur. The first is the normal pairwise interaction that results between atoms 1 through n in the system. The other two types of interactions result from the wraparound effect that occurs when an atom within the atomic chain interacts with an atom that lies in a periodic image. For each atom, there are forward and backward wraparound interactions with atoms lying in periodic images of the chain in the positive and negative directions respectively. These three types of periodic interactions are shown in Figure 2.10.

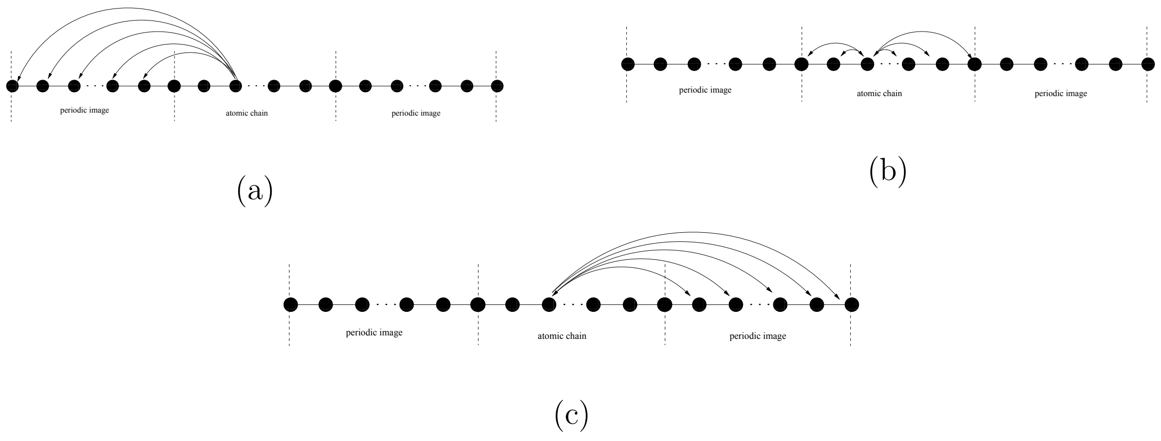


Figure 2.10: Graphs of (a) backwards wraparound interactions, (b) normal pairwise interactions, and (c) forward wraparound interactions for an atom in the chain of atoms

The main difficulty in the implementation of periodic boundaries is identifying each type of interaction, and how many interactions should be accounted for. Since there is are infinitely many atomic interactions, we use the cutoff distance to limit the number of atomic interactions in the system. For each individual atom, i, we loop through atoms both forward and backward to find all atoms that are separated by less than r_c. While doing so, the interaction can be classified as either a forward interaction or backward interaction. Then the forces acting upon atom i can be calculated accordingly. If the force is resulting from a forward interaction, then the force acting upon atom i is a negative force. If the force is resulting from a backward interaction, then the force acting upon i is positive.

For this boundary condition, the ode function passed into ode45 is LJODE_ periodic.m which, given the neighbor list constructed by periodicNeighborList.m, identifies the normal pairwise interactions, forward wraparound interactions, and backwards wraparound interactions. The forces resulting from each of these interactions are then calculated accordingly. As in the other simulations, the construction is based upon the number of neighbor list reconstructions over the entire runtime, as well as the cutoff distance, r_c.

The same system of atoms, with initial positions as given in Table 2.2, is utilized to observe the effects of the periodic boundary condition. This time, the atoms are expected to oscillate, and eventually equilibrate at uniform separation distances across the system. The graph given in Figure 2.11 provides the positions of atoms over time with periodic boundaries.

As seen in the figure, the atoms oscillate slightly until they equilibrate at uniform pairwise separation distances. As expected, the periodic boundaries confine the chain of atoms to a defined simulation space without imposing physical boundaries. We also note that the periodic boundaries appear to result in uniform spacing in the equilibration configuration of atoms. This differs from the other boundary conditions. Also, the final equilibrium configuration is independent of the placement of physical boundaries as in the walls or stationary end atoms cases. The periodic nature of most systems of atoms make periodic boundaries a physically relevant boundary con-

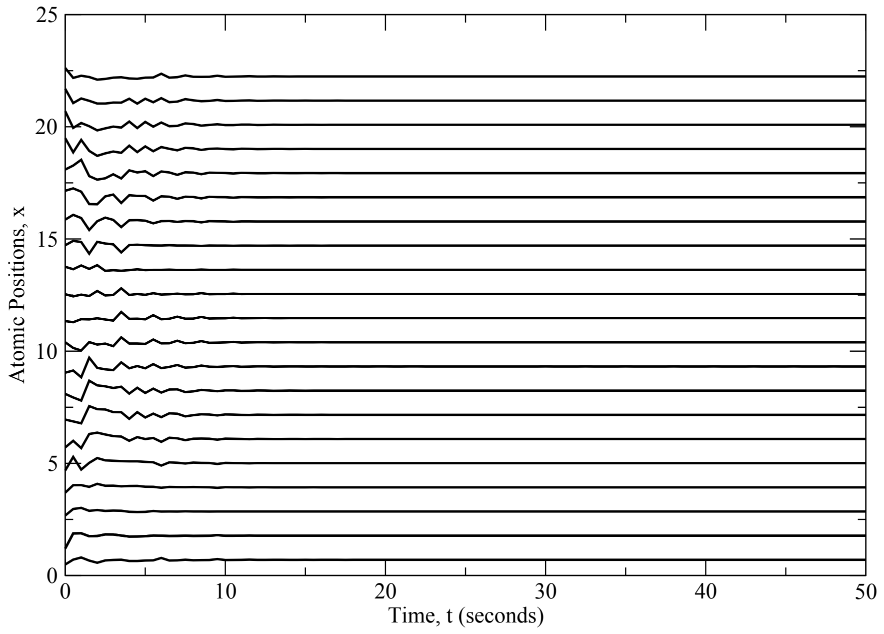


Figure 2.11: Position vs time graph of a system of $n = 21$ atoms with periodic boundaries

dition. It also represents the behavior of a chain wrapped around a cylinder or, in two dimensions, a lattice rolled into a tube, much like rolling a graphene lattice into a CNT. For this reason, modeling a graphene sheet with periodic boundaries is a possible method for modeling large CNTs.

2.7 Summary of Results

A one-dimensional chain of atoms is modeled using MD simulation to simulate the motion of atoms over time. A system of first-order ordinary differential equations is developed based on Newton's second law. Although the Lennard-Jones potential is used in simulations, the code is structured so that another potential function can easily replace it to model other types of atoms within a chain. Using fixed end atoms, walls, and periodic boundaries, the effects of various boundary conditions are compared. Table 2.5 contains the separation distances between adjacent atoms at their final positions for each set of boundary conditions. The boundary conditions that result in the most uniform separation distances, as seen in Table 2.5, is the periodic boundary conditions, suggesting that periodic boundaries are suitable when focusing on the behavior of the interior atoms in a large system.

Next we consider more physically relevant MD simulations of two-dimensional systems of atoms. The same three types of boundary conditions are incorporated into the two-dimensional model. Further, defects in an LJ-type lattice are simulated.

Table 2.5: Final distances between atoms (in angstrom, Å) for the test case under various boundary conditions

Atoms	No BC	Fixed	Walls, k=1	Walls, k=.5	Walls, k=0	Periodic
1, 2	15.77931	1.10776	7.55639	9.61281	11.66841	1.07721
2, 3	1.12092	1.10647	1.12092	1.12092	1.12092	1.07719
3, 4	1.11942	1.10639	1.11942	1.11942	1.11942	1.07721
4, 5	1.11931	1.10637	1.11932	1.11932	1.11932	1.07719
5, 6	1.11930	1.10637	1.11930	1.11930	1.11930	1.07721
6, 7	1.11929	1.10637	1.11930	1.11930	1.11930	1.07719
7, 8	1.11929	1.10637	1.11930	1.11930	1.11930	1.07721
8, 9	1.11929	1.10637	1.11930	1.11930	1.11930	1.07722
9, 10	1.11929	1.10637	1.11930	1.11930	1.11930	1.07718
10, 11	1.11929	1.10637	1.11930	1.11930	1.11930	1.07722
11, 12	1.11929	1.10637	1.11930	1.11930	1.11930	1.07719
12, 13	1.11929	1.10637	1.11930	1.11930	1.11930	1.07718
13, 14	1.11929	1.10637	1.11930	1.11930	1.11930	1.07722
14, 15	1.11929	1.10637	1.11930	1.11930	1.11930	1.07718
15, 16	1.11929	1.10637	1.11930	1.11930	1.11930	1.07721
16, 17	1.11930	1.10637	1.11930	1.11930	1.11930	1.07719
17, 18	1.11931	1.10637	1.11932	1.11932	1.11932	1.07720
18, 19	1.11942	1.10639	1.11942	1.11942	1.11942	1.07720
19, 20	1.12092	1.10647	1.12092	1.12092	1.12092	1.07720
20, 21	10.03670	1.10776	6.31848	7.25059	8.17975	1.07720

CHAPTER III

TWO-DIMENSIONAL SIMULATIONS

Next, we add a second dimension to the one-dimensional model in order to conduct two-dimensional MD simulations. In two dimensions, the system of atoms is no longer a linear chain of atoms, but a lattice in which the position of each is defined by a vector in $\mathbf{R}^2$. We use classical MD simulation to investigate the behavior of lattice atoms governed by the Lennard-Jones potential. The three boundary conditions investigated in the one-dimensional model are extended to the two-dimensional case. First, the model of a lattice of atoms is developed. Then, the corresponding numerical model is formulated and solved in two dimensions. As before, all code is written and executed in MATLAB and is provided in Appendix B.

3.1 Two-Dimensional Model Formulation

The two-dimensional system of atoms is initialized as a triangular lattice with n atoms arranged in c columns and r rows. The angle between the basis vectors of the lattice is given by θ. For an LJ-type lattice, $\theta = 60°$— the angle we use throughout the simulations. The lattice is also constructed so that the distance between every pair of adjacent atoms is uniform throughout. The uniform separation distance can be varied by simulation, and random nonuniform perturbations may be applied to positions and/or velocities of individual atoms. Figure 3.1 illustrates the two-dimensional lattice construction.

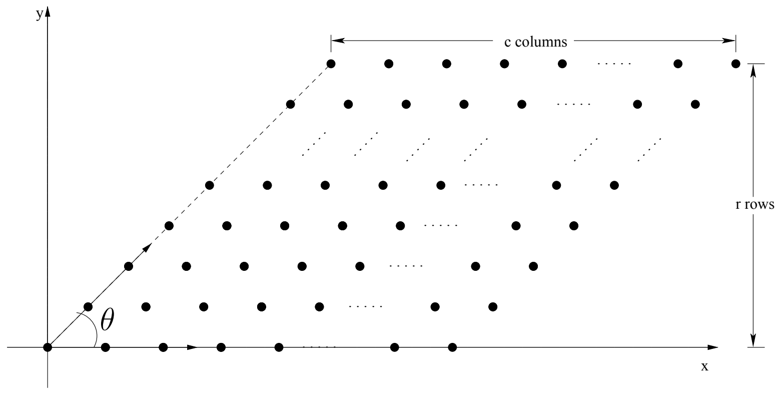


Figure 3.1: Diagram of a two-dimensional atomic lattice

The governing differential equation for classical MD simulations, as given in Chapter 2, is the second order differential equation for damped motion and is given by

$$m\ddot{\mathbf{x}} + \beta\dot{\mathbf{x}} = \mathbf{F}, \tag{3.1}$$

where $\mathbf{F}_i$ is the force resulting from LJ-type interactions. This corresponds to equation (2.5) in one-dimension, but now the positions of atoms are defined by two-dimensional vectors. Force is also now defined as a two-dimensional vector, such that

$$\mathbf{F}_i = -\nabla_{\mathbf{x}_i} \sum_{i \neq j} U(r_{ij}), \tag{3.2}$$

where $\mathbf{F}$ is force, U is potential, and i is the index of a given atom with position $\mathbf{x}_i$. Furthermore,

$$r_{ij} = |\mathbf{x}_i - \mathbf{x}_j| \tag{3.3}$$

is the distance between atoms i and j. Similar to force, the positions of atoms are now given by two-dimensional vectors, $\mathbf{x}_i = (x_{i_1}, x_{i_2})$, for atom i. Similarly, the velocity of an atom is given by $\mathbf{v}_i = \langle v_{i_1}, v_{i_2} \rangle = \sqrt{{v_{i_1}}^2 + {v_{i_2}}^2}$. In two dimensions, force can no longer be considered a simple derivative of the potential, but rather a two-dimensional spatial gradient, as shown in equation (3.2). By the chain rule,

$$\nabla_{\mathbf{x}_i} U(|\mathbf{x}_i - \mathbf{x}_j|) = U'(|\mathbf{x}_i - \mathbf{x}_j|)\nabla|\mathbf{x}_i - \mathbf{x}_j|. \tag{3.4}$$

By letting

$$|\mathbf{x}_i - \mathbf{x}_j|^2 = (x_{i_1} - x_{j_1})^2 + (x_{i_2} - x_{j_2})^2, \tag{3.5}$$

the gradient of this term is

$$\nabla_{\mathbf{x}_i} |\mathbf{x}_i - \mathbf{x}_j|^2 = 2(\mathbf{x}_i - \mathbf{x}_j). \tag{3.6}$$

Also by the chain rule,

$$\nabla_{\mathbf{x}_i} |\mathbf{x}_i - \mathbf{x}_j|^2 = 2|\mathbf{x}_i - \mathbf{x}_j|\nabla|\mathbf{x}_i - \mathbf{x}_j|. \tag{3.7}$$

Given equations (3.6) and (3.7), the gradient of the distance between two atoms is given by the unit vector,

$$\nabla_{\mathbf{x}_i} |\mathbf{x}_i - \mathbf{x}_j| = \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}. \tag{3.8}$$

Using this, the force acting upon atom i from atom j is given by

$$\mathbf{F}_{ij} = -U(|\mathbf{x}_i - \mathbf{x}_j|)\frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}. \tag{3.9}$$

Also note that the force acting upon atom j from atom i is $\mathbf{F}_{ji} = -\mathbf{F}_{ij}$. The Lennard-Jones potential is used to model the pairwise interactions between atoms throughout the system so that the force between two atoms is then given by

$$\mathbf{F}_{ij} = -24\epsilon \left[2\frac{\sigma^{12}}{|\mathbf{x}_i - \mathbf{x}_j|^{13}} - \frac{\sigma^6}{|\mathbf{x}_i - \mathbf{x}_j|^7} \right] \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}. \tag{3.10}$$

The system of governing differential equations of the two-dimensional MD simulation is given by

$$m\ddot{\mathbf{x}}_i + \beta\dot{\mathbf{x}}_i = -\sum_{j=1;j\neq i}^{n} 24\epsilon \left[2\frac{\sigma^{12}}{|\mathbf{x}_i - \mathbf{x}_j|^{13}} - \frac{\sigma^6}{|\mathbf{x}_i - \mathbf{x}_j|^7} \right] \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}. \tag{3.11}$$

This system is solved numerically as we discuss in the next section.

3.2 Numerical Model in Two Dimensions

Although unit vectors must be calculated for all pairs of atoms at every time step, the numerical setup is similar to that in one dimension. Letting the governing system of equations in (3.11) be written as a system of first order equations,

$$\begin{aligned} \frac{\partial \mathbf{x}_i}{\partial t} &= \mathbf{v}_i \\ \frac{\partial \mathbf{v}_i}{\partial t} &= \frac{1}{m}\mathbf{F}_i - \frac{\beta}{m}\mathbf{v}_i, \end{aligned} \tag{3.12}$$

such that $\mathbf{F}_i$ defines the force according to Lennard-Jones in two dimensions as given in equation (3.10).

Recall that, in two dimensions, n defines the number of atoms, c denotes the number of columns, r defines the number of rows, and $\theta = 60°$ defines the angle between the basis vectors of the lattice. Thus if $c = 100$ and $r = 100$, the atomic system is initialized as a 100 by 100 lattice, with $n = 10000$ atoms. The lattice is initialized so that the spacing between adjacent atoms is uniform, and equal to εr_{eq}. Here $r_{eq} = 2^{(1/6)}$ Å and ε defines uniform perturbations of the equilibrium spacing. If $\varepsilon < 1$, the uniform spacing is less than the equilibrium spacing and the atoms will be repelled initially. If $\varepsilon > 1$, the uniform spacing is greater than equilibrium spacing and the atoms will be attracted initially. If $\varepsilon = 1$, then the uniform spacing is approximately equal to that in equilibrium and the atoms will have little to no movement. For larger numbers of atoms, the equilibrium distance between atoms, r_{eq}, will become slightly less than $2^{(1/6)}$ Å. This is because $r_{eq} = 2^{(1/6)}$ Å is derived assuming a pairwise interaction for two atoms. However, as an atomic system becomes larger, the value for r_{eq} decreases slightly due to interaction with multiple neighbors.

Next, the behavior of atoms within the atomic lattice is investigated using MATLAB. First, we initialize the atomic structure and numerically solve the system of equations by utilizing the ode45 MATLAB routine. The code written for the two-dimensional study is similar to that in the one-dimensional study. The main file is TwoD_MDSystem.m, which first defines all necessary parameters, then initializes the atomic lattice by prescribing initial positions and velocities for each atom in the system. After all of the initializations are executed, TwoD_MDSystem.m calls the ode45 solver, passing in an ode function to numerically solve the system of equations. As in the one-dimensional study, the ode function varies based on the boundary conditions imposed upon the system. The ode functions used for no boundaries, fixed boundaries, walls, and periodic boundaries are named LJODE_NoBC.m, LJODE_Fixed.m, LJODE_Walls.m, and LJODE_Periodic.m, respectively.

While the implementations of ode functions varies between boundary conditions, the overall algorithm is the same. The system of first order differential equations (3.13) is solved numerically over a given time interval. The function first constructs a neighbor list for the atomic lattice by calling NeighborList.m, then loops through the neighbor list in order to calculate the forces between each pair of atoms stored in the neighbor list vector. The distance between two atoms is calculated by calling the CalculateDistance.m function. Once the distance is calculated, the unit vectors between pairs of atoms are computed, along with forces between pairs of atoms in the neighbor list by calling the function file LJForce.m. The resulting force upon each atom is a sum of the forces from its atomic interactions stored in the neighbor list. This final net force is stored in two vectors, NetFx and NetFy, which contain the x and y components of the force acting upon each atom. This process is repeated for a discrete number of timesteps until the final runtime is reached. In this chapter we utilize neighbor list construction throughout, so next we discuss neighbor list construction in two dimensions.

3.3 Neighbor List in Two Dimensions

In two dimensions, the construction of the neighbor list is slightly different, due to the additional dimension. As in one dimension, the neighbor list is used to construct a two-column vector, storing pairs of atoms with interatomic distances less than a given cutoff radius. The potential is thus

$$U(r) = \begin{cases} -24\epsilon \left[2\frac{\sigma^{12}}{r^{13}} - \frac{\sigma^6}{r^7} \right], & r \leq r_c \\ 0, & r > r_c \end{cases} \tag{3.13}$$

so that the force between pairs of atoms is of Lennard-Jones type as long as the distance between the atoms is less than the prescribed cutoff radius, r_c. If the distance is greater than r_c, the potential and resulting force between them is 0. In one dimension, the distance between two atoms, atom i, and atom j, is calculated by

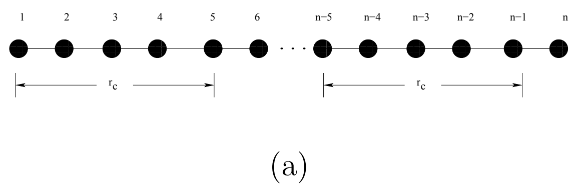
$$r_{ij} = x_i - x_j, \tag{3.14}$$

where x_i and x_j are the positions of the i and j atoms respectively. In two dimensions, the distance between two atoms is now

$$r_{ij} = |\mathbf{x}_i - \mathbf{x}_j| = \sqrt{(x_{i_1} - x_{j_1})^2 + (x_{i_2} - x_{j_2})^2}, \tag{3.15}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the positions of atoms i and j. The cutoff radius given by r_c defines a disk with radius r_c around every atom. The atom at the center of the circle then interacts with every other atom the lies inside the circle (Figure 3.2).

As in Chapter 2, the cutoff radius is a parameter that increases the speed of simulation on the expense of decreased accuracy. We showed in Chapter 2 that $r_c = 5$ Å is the optimal cutoff value, since it results in small error and a much faster runtime. In two dimensions we again use $r_c = 5$ Å, as we conjecture that there should be no difference between the values of the cutoff radius in one and two dimensions. To continue with this two-dimensional model, we model lattices dominated by repulsive



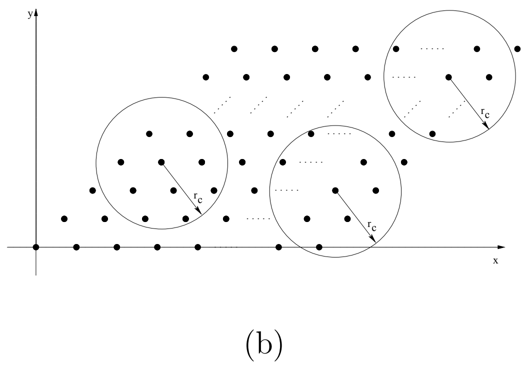


Figure 3.2: Figures demonstrating neighbor list construction in (a) one dimension and (b) two dimensions

and attractive behaviors, demonstrating the need for boundary conditions to confine atoms to a finite domain.

3.4 Example Two-Dimensional Simulations

To model the behavior of a lattice dominated by repulsive and attractive behaviors, we simulate a 5 by 5 system of 25 atoms, oriented in 5 rows and 5 columns, with uniform perturbations of interatomic distances given by $\varepsilon = .8$, and $\varepsilon = 1.2$, respectively. Furthermore, we simulate a similar lattice for $\varepsilon = 1$ in order to observe the behavior of atoms when the initial configuration is near equilibrium. The system parameters are given in Table 3.1.

The 5 by 5 lattice is first constructed with perturbation parameter $\varepsilon = .8$, so that the uniform spacing is $.8r_{eq}$. Since $.8r_{eq} < r_{eq}$, it is expected that the atoms will initially repel, reaching some equilibrium configuration given sufficient time. The initial position plot of the atomic system is provided in in Figure 3.3, along with the

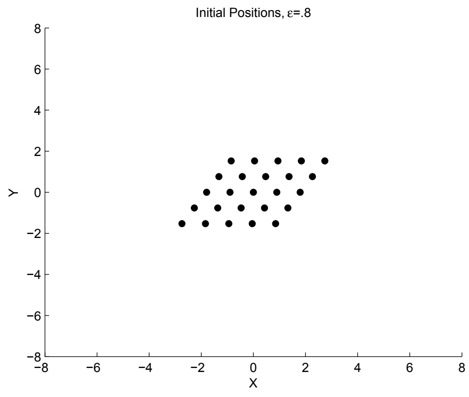
Table 3.1: Variables and parameters used in the 5 by 5 atomic lattice

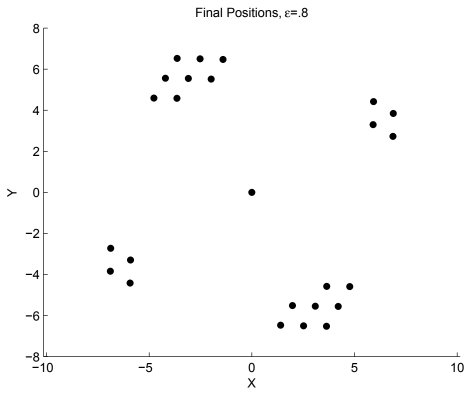
Parameter	Description	Value
n	Number of atoms	25 atoms
r	Number of rows	5
c	Number of columns	5
m	Atomic mass	1.0 amu
σ	Separation at zero potential	1.0 Å
ϵ	Well depth, (strength of interaction)	1.0 kJ/mol
β	Damping coefficient, (strength of damping)	1.0 eV(ps/Å)

final position plot. The phase space plot that graphs the x and y coordinates of atoms over time is also provided.

As seen in Figure 3.3, indeed the atoms are initially repelled. Subsequently, they settle in a configuration where they no longer experience repulsive forces. This configuration consists of several small groups of atoms. Since the groups of atoms are each separated by large distances compared to lattice spacing, it is likely that they will no longer be attracted to each other since the forces between them are zero due to the introduction of the cutoff distance. Thus, the atoms equilibrate into a configuration in which the lattice is broken.

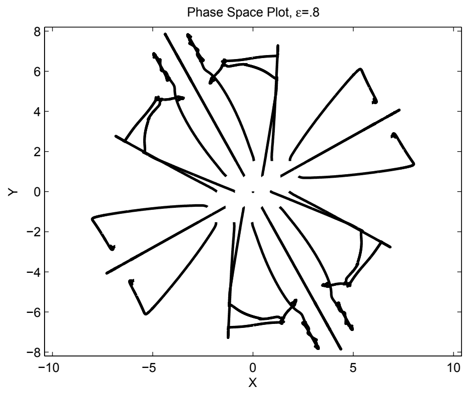
The next simulation executed is a system with $\varepsilon = 1$, so that the initial uniform spacing is r_{eq}. Because there are more than two atoms in the system, the actual equilibrium distance is slightly less than $r_{eq} = 2^{(1/6)}$ Å. Therefore, it is conjectured that very slight inward movement will occur in the system, as atoms move slightly





(a)

(b)

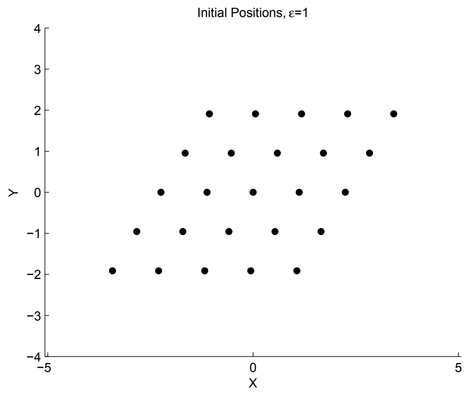


(c)

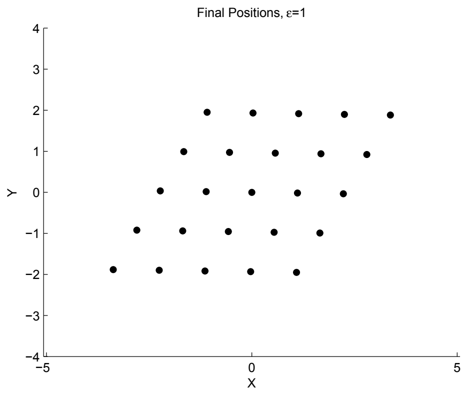
Figure 3.3: (a) Initial positions, (b) final positions, and (c) phase space plot graphs for a two-dimensional atomic lattice with initial spacing of $.8r_{eq}$

closer together before settling into equilibrium configuration. The initial positions, final positions, and phase space plots provided in Figure 3.4 show this behavior of the nearly equilibrated lattice.

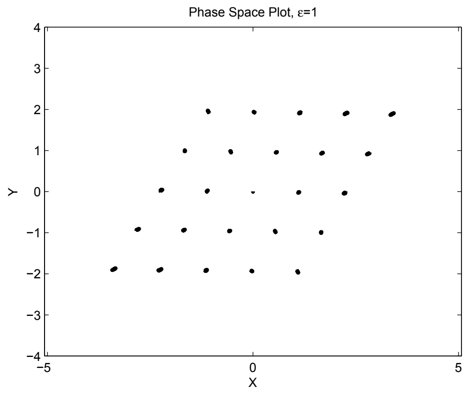
As expected, given initial spacings of r_{eq}, relatively small interatomic forces result, leading to little to no movement throughout the system. As shown in Figure



(a)



(b)

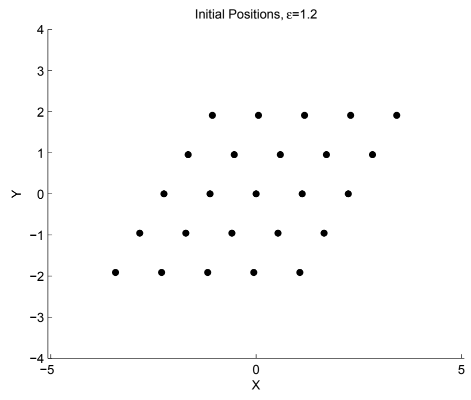


(c)

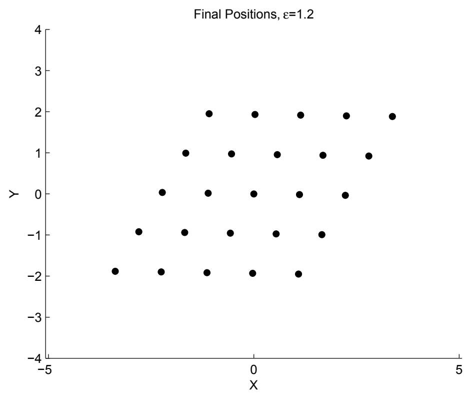
Figure 3.4: (a) Initial positions, (b) final positions, and (c) phase space plot graphs for a two-dimensional atomic lattice with initial spacing of r_{eq}

3.4, the atoms move very slightly from their initial positions before the system equilibrates.

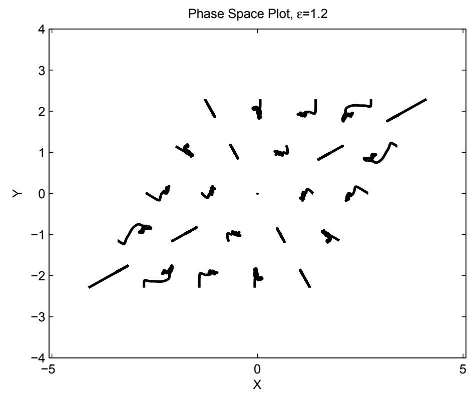
Finally, a system with initial spacing greater than r_{eq} is simulated to observe attractive behavior in the atomic lattice. The uniform perturbation parameter is set to be $\varepsilon = 1.2$ so that the uniform spacing is $1.2r_{eq}$. With this initial spacing, the atoms are expected to gradually attract to each other until they settle into equilibrium positions. The initial position plot, final position plot, and phase space plot are provided in Figure 3.5.



(a)



(b)



(c)

Figure 3.5: (a) Initial positions, (b) final positions, and (c) phase space plot graphs for a two-dimensional atomic lattice with initial spacing of $1.2r_{eq}$

As supported by Figure 3.5, when the initial uniform spacing is greater than r_{eq}, the atoms are initially attracted to each other. The attractive forces dominate until the atoms equilibrate at distances approximately equal to r_{eq}. Once this equilibrium state is reached, there is no more motion within the system.

Various boundary conditions can now be imposed upon the system in order to keep all atoms in the system within a confined space. The three boundary conditions to be studied in two dimensions are: stationary boundary atoms, walls, and periodic boundary conditions. The resulting behavior of the two-dimensional lattice is observed under each of the boundary conditions.

3.5 Boundary Condition: Stationary Boundary Atoms

The first set of boundary conditions is the fixed case, in which boundary atoms are held in fixed positions over time. In one dimension, this case corresponds to two fixed atoms at the end of the atomic chain. In two dimensions, the number of stationary atoms in the system is no longer a fixed number, as it now varies depending on the size of the atomic lattice. Here, every atom on each of the four sides of the lattice is held fixed throughout the simulation, allowing all other atoms to move freely. This construction is shown in Figure 3.6, in which the stationary boundary atoms are outlined on the edges of the lattice.

The number of stationary atoms is dependent upon the number of rows, r, and number of columns, c, so that $2c + 2r - 4$ atoms are held fixed. All other atoms within the system are able to move freely in the space defined by the fixed atoms. It is possible that moving atoms could penetrate the boundary with large enough initial velocity, but here we assume zero initial velocity and near equilibrium initial lattice

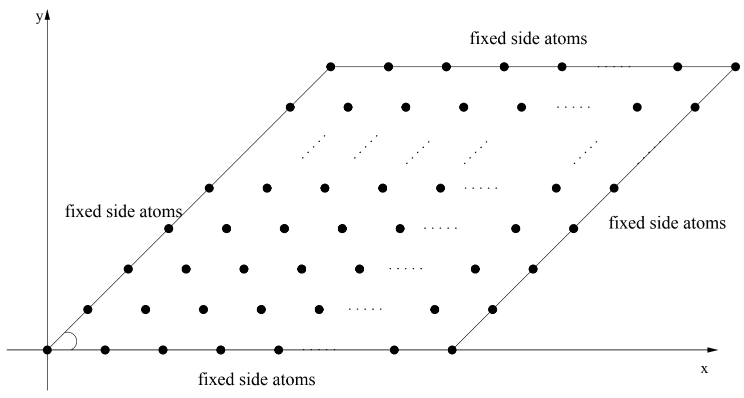


Figure 3.6: Diagram of the two-dimensional lattice with fixed boundary atoms

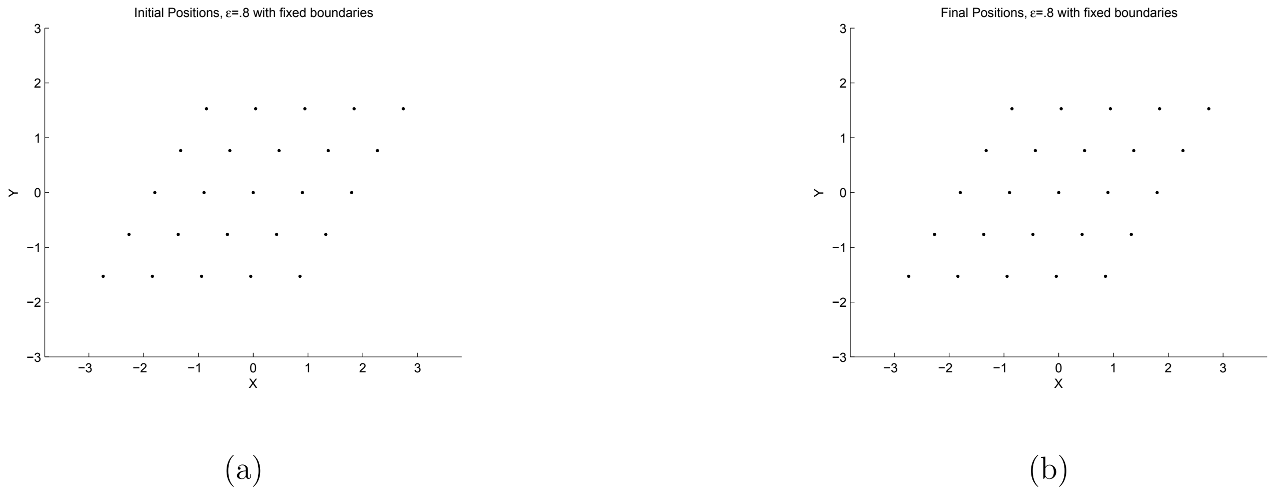
so that all atoms remain inside. While there are no forces on fixed atoms, freely moving atoms have forces acting upon them resulting from pairwise interactions with each other and with the fixed atoms. The fixed atoms in the system are initialized with zero initial velocity, so that with zero net force they remain stationary over time. This is equivalent to solving $2n - (2c + 2r - 4)$ equations at every time step. The ode function for this boundary condition is LJODE_Fixed.m, which calculates atomic forces based on the indices of atoms that are stored in the neighbor list. Recall that each row in the neighbor list corresponds to an atomic interaction. A vector storing the indices of each of the stationary atoms is used to identify fixed atoms during calculations so that the forces acting upon these atoms are ignored.

To test this boundary condition, the 5 by 5 atomic lattice with $.8r_{eq}$ uniform initial spacing is used. In the simulation of this system without boundary conditions, all atoms repel each other immediately, since the initial spacing is less than r_{eq}, resulting in large repulsive forces. With this boundary condition, there are now 16 stationary atoms surrounding the edge of the lattice, leaving 9 atoms to move freely throughout the region surrounded by stationary atoms. It is expected that the 9 freely moving atoms move minimally throughout the simulation, since their movement is confined to this region. Note that if the uniform spacing is much smaller— e.g. $.2r_{eq}$— atoms may penetrate the boundary due to very large repulsive forces. The initial position plot, final position plot, and phase space plot of this simulation, ran until a final time of $t_f = 1000$ seconds, are each provided in Figure 3.7.

As shown in the figure, the fixed atoms indeed restrict movement of the freely moving atoms by confining them inside of the boundary. The phase space plot in this case has no lines resulting from the movement of atoms, showing that there is little to no movement for all freely moving atoms in the system. It is also observed that, after a runtime of $t_f = 1000$ seconds, the center atom has not yet reached an equilibrium position, indicating that the uniform spacing observed between atoms might not be maintained in the equilibrium stressed configuration. Next we explore lattice constrained by the ‘walls’

3.6 Boundary Condition: Walls

In one dimension, fixed ‘walls’ are placed on either side of the atomic chain so that only atoms 1 and n are able to collide with the wall. In two dimensions this case is significantly different because walls are now placed on vertical and horizontal boundaries, forming a box that surrounds the lattice. The positions of each of the



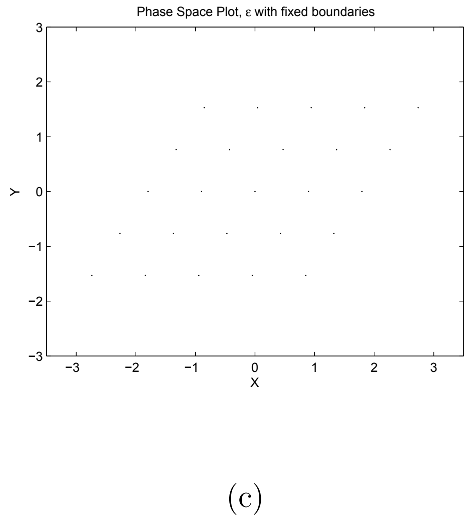


Figure 3.7: (a) Initial positions, (b) final positions, and (c) phase space plot for a lattice with initial uniform spacing $.8r_{eq}$ with stationary boundary atoms

four walls may be varied between simulations, but they are held fixed during a given simulation. What is also different in two dimensions is that every atom in the lattice can collide with any of the four walls at any time, where before only the two atoms at the end of the one-dimensional chain can collide with the walls. The atomic lattice is initialized with uniform spacing and surrounded by the fixed walls (Figure 3.8).

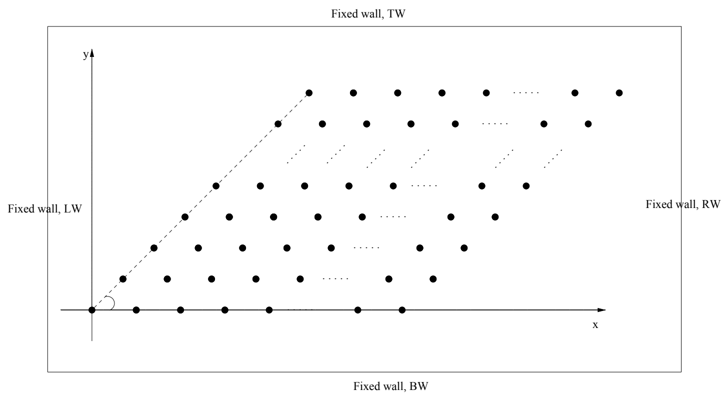


Figure 3.8: Diagram of the two-dimensional lattice with 'walls'

Since any atom can hit any of the walls at any time, we must prescribe a procedure to determine the post interaction velocity of an atom after it collides with a wall. In the one-dimensional study, a parameter k is introduced to define the hardness of the walls. The same parameter is used in two dimensions for each wall so that, if $k = 0$, the walls are soft and an atom will have zero velocity after contact. When $k = 1$ in the one-dimensional study, the walls are assumed to be perfectly hard and atoms colliding with a wall at velocity $\mathbf{v}$ bounced off with velocity $-\mathbf{v}$. In two dimensions, if an atom collides with either the left or right walls, its horizontal velocity component is calculated to be $-k$ times the horizontal component of velocity prior to the collision, while the vertical component of velocity remains the same. Thus,

$$v_{i1_f} = -k v_{i1_0} \tag{3.16}$$

$$v_{i2_f} = v_{i2_0} \tag{3.17}$$

where the subscript 0 denotes the velocity of atom i before it hits the wall, and the subscript f denotes its velocity after it hits the wall. If an atom collides with the top or bottom wall, the horizontal component of its velocity stays the same, but the vertical component of velocity is equal to $-k$ times the vertical component of velocity prior to collision. Therefore,

$$v_{i_{1_f}} = v_{i_{1_0}} \tag{3.18}$$

$$v_{i_{2_f}} = -kv_{i_{2_0}} \tag{3.19}$$

The positions of the atom immediately after a collision is adjusted by $WA = 10^{-12}$ Å inward. As in one dimension, this adjustment has no physical relevance, but it allows to avoid the occurrence of infinite loops in the events function calls by ode45 without sacrificing accuracy. If an atom hits any of the four walls, its x and y coordinates are modified as follows:

$$x_{i_f} = x_{LW} + WA \tag{3.20}$$

$$y_{i_f} = y_{i_0} \tag{3.21}$$

for the left wall,

$$x_{i_f} = x_{RW} - WA \tag{3.22}$$

$$y_{i_f} = y_{i_0} \tag{3.23}$$

for the right wall,

$$x_{i_f} = x_{i_0} \tag{3.24}$$

$$y_{i_f} = y_{TW} - WA \tag{3.25}$$

for the top wall, and

$$x_{i_f} = x_{i_o} \tag{3.26}$$

$$y_{i_f} = y_{BW} + WA \tag{3.27}$$

for the bottom wall. Again, the subscript 0 denotes the position of an atom before it hits the wall, and f denotes its position after the collision. The parameters x_{LW}, x_{RW}, y_{TW}, and y_{BW} denote the positions of the left, right, top, and bottom walls.

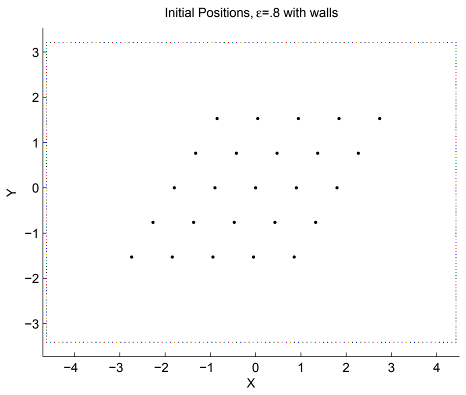
The ode function for this boundary condition, LJODE_Walls.m, is called by the main routine after initializing the positions of atoms that form the lattice, along with other necessary parameters. This function numerically solves the $2n$ equations by utilizing the MATLAB ode45 routine. In order to detect if an atom collides with a wall, the MATLAB events function calls the user-provided function, wallsEvents.m, which checks the positions of atoms at every time step to identify if any of the atoms have crossed any of the walls. If an atom collides with a wall, ode45 exits and the code execution returns to the main routine. Then, the position of the atom in contact with the wall is adjusted according to (3.20) - (3.27). The velocity of the atom is calculated according to which wall it hits, as shown by (3.16) - (3.19). Then, timestep.m is called to calculate the next time step to be passed back into ode45, along with the current solution as the new initial vector. This process is repeated every time an atom collides with a wall until the final time, t_f, is reached.

By surrounding the lattice with walls, the atoms are forced to remain inside the surrounding box. The influence of the walls on the equilibrium configuration of atoms is studied by placing fixed walls around the 5 by 5 lattice with uniform initial spacing of $.8r_{eq}$. With no boundaries, the atoms in the system are ejected from their initial positions due to strong repulsive force. As a result the atoms spread out among a much larger domain. By using walls, the 25 atoms now remain inside the surrounding box. The initial positions plot, final positions plot, and phase space plot for this simulation are provided in Figure 3.9.

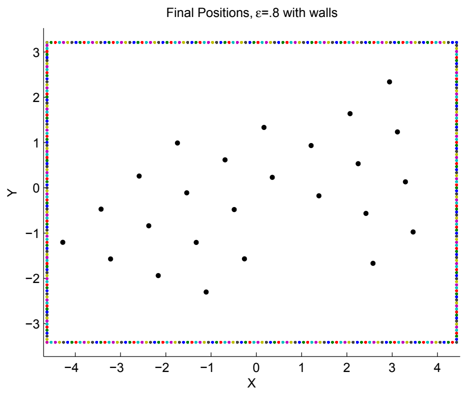
As shown in the phase space diagram, many of the atoms in the lattice bounce off the walls after the initial repulsion. It is also observed that the same atom may bounce off the same wall multiple times or bounce off many different walls. As shown in the final positions plot, after atoms repel and bounce off of the walls, they aggregate into a stable configuration in which all atoms are separated from adjacent atoms by approximately r_{eq}. The outcome of simulations depends upon the positions and hardness of the walls. If a larger box surrounds the lattice, then it is likely that the atoms would not equilibrate with uniform spacing.

3.7 Periodic Boundary Conditions

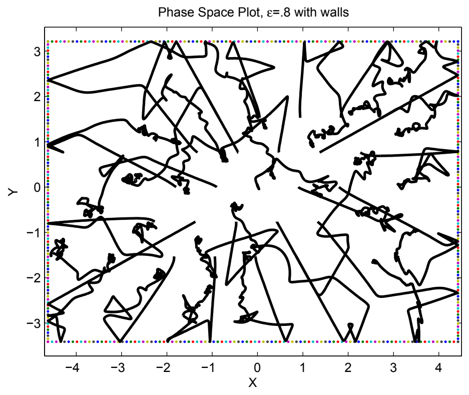
The final boundary condition that we study is the periodic boundary condition. In one dimension, the periodic boundary condition is implemented by considering periodic copies of the atomic chain in both the positive and negative directions of the x-axis. In two dimensions, periodic boundaries are implemented in both vertical and horizontal directions, as if there is an infinite number of periodic copies of the atomic lattice in every direction (Figure 3.10).



(a)



(b)



(c)

Figure 3.9: (a) Initial positions, (b) final positions, and (c) phase space plot for a lattice with initial uniform spacing $.8r_{eq}$ with walls

First, the two-dimensional lattice is constructed so that it is centered at the origin. Two new parameters, L_x and L_y, are prescribed to define the width of the periodic box surrounding the lattice in the x- and y-directions, respectively. The periodic box is then oriented around the atomic lattice between $x = -\frac{L_x}{2}$ and $x = \frac{L_x}{2}$ in the x-direction, and $y = -\frac{L_y}{2}$ and $y = \frac{L_y}{2}$ in y-direction, as shown in Figure 3.11.

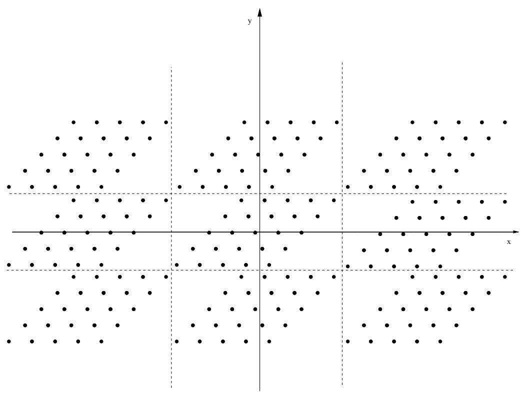


Figure 3.10: Diagram of the two-dimensional atomic lattice with periodic boundaries

Using this periodic box, the wraparound effect from periodicity must be considered in the calculations of interatomic forces at every time step. Also, the position of every atom at every time step must be examined in order to ensure that it remains within the periodic box. Because of periodicity, if an atom leaves the periodic region through the $x = \frac{L_x}{2}$ boundary, its x-coordinate should be adjusted to $x = -\frac{L_x}{2}$, and if it moves outside $x = -\frac{L_x}{2}$, it is repositioned at $x = \frac{L_x}{2}$. Similarly, in the y direction, if an atom leaves the periodic region at $y = \frac{L_y}{2}$, its position is set to $y = -\frac{L_y}{2}$ and vice versa. Mathematically, this is governed by the following calculations.

If $x_{i_0} > \frac{L_x}{2}$, then

$$x_{i_f} = x_{i_0} - L_x \tag{3.28}$$

$$y_{i_f} = y_{i_o}, \tag{3.29}$$

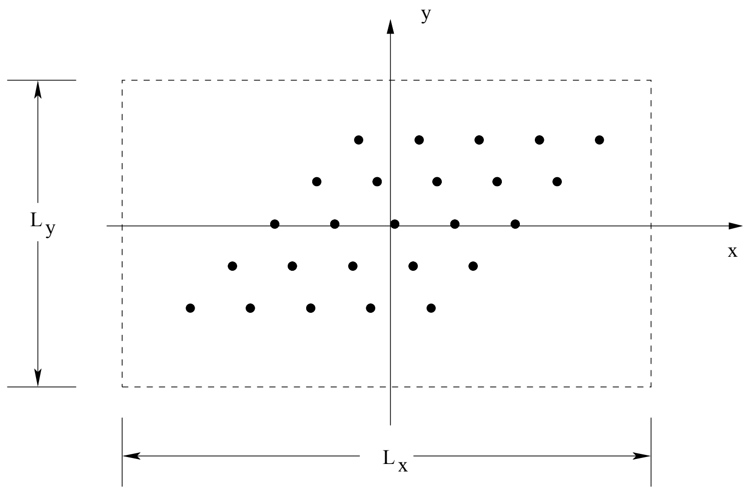


Figure 3.11: Diagram illustrating the periodic box with the atomic lattice centered at the origin

and if $x_{i_0} < -\frac{L_x}{2}$, then

$$x_{i_f} = x_{i_0} + L_x \tag{3.30}$$

$$y_{i_f} = y_{i_0}. \tag{3.31}$$

Similarly, if $x_{i_0} > \frac{L_y}{2}$, then

$$x_{i_f} = x_{i_0} \tag{3.32}$$

$$y_{i_f} = y_{i_0} - L_y, \tag{3.33}$$

and if $y_{i_0} < -\frac{L_y}{2}$, then

$$x_{i_f} = x_{i_0} \tag{3.34}$$

$$y_{i_f} = y_{i_0} + L_y. \tag{3.35}$$

In the equations, the subscript 0 denotes the position of an atom before adjustment and f denotes the position after the atom is repositioned.

After ensuring that all atoms remain within the periodic region, the necessary interactions may be calculated in order to determine the forces acting on each atom in the system. In what follows, we assume that the cutoff radius of LJ-type interactions does not exceed the smallest dimension of the periodic box. Then, the interactions are found by considering four cases which account for the periodic wraparound effect. If the difference $x_j - x_i$ exceeds $\frac{L_x}{2}$, forward wraparound in the x-direction occurs. This means that, if the x-component of distance between the atoms i and j atoms is greater than $\frac{L_x}{2}$, atom i instead interacts with the right periodic image of atom j. If the difference between x_i and x_j is less than $-\frac{L_x}{2}$, backward wraparound occurs and atom i interacts with the left periodic image of atom j. Likewise, in the y-direction, the upward or downward shift of the atom j is executed wherever $y_j - y_i$ is greater than $\frac{L_y}{2}$ or is less than $-\frac{L_y}{2}$, respectively. Mathematically, let

$$dx = x_i - x_j, \tag{3.36}$$

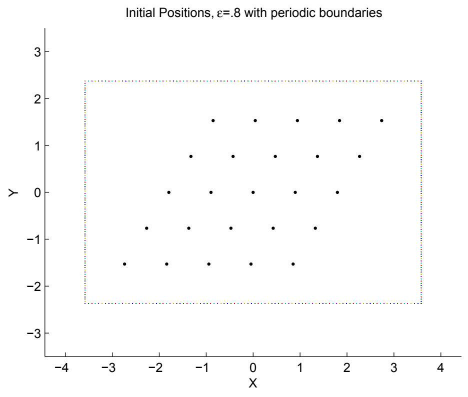
so that dx is the difference between the x-coordinates of atoms i and j. If $dx \geq \frac{L_x}{2}$, forward wraparound occurs by defining $dx = dx - L_x$. If $dx < -\frac{L_x}{2}$, backward wraparound occurs by letting $dx = dx + L_x$. Similarly, in the y-direction,

$$dy = y_i - y_j. \tag{3.37}$$

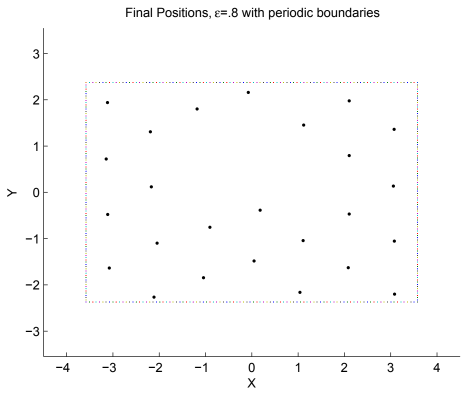
If $dy \geq \frac{L_y}{2}$, forward wraparound occurs and $dy = dy - L_y$. If $dy < -\frac{L_y}{2}$, backward wraparound occurs and $dy = dy + L_y$.

The ode function used for periodic boundaries is LJODE_Periodic.m. Within this file, the interactions between atoms are all calculated as a subprocess of ode45 while solving the $2n$ equations. Within the same function, the periodic wraparound effect is taken into account by computing dx and dy for pairs of atoms in the neighbor list at the current time step; then periodic wraparound forces are determined accordingly. In order to ensure that atoms remain inside the periodic region, the MATLAB ode45 events function is used, defining a new function, periodicEvents. This checks at every time step whether the atoms remain inside the periodic box, to identify any atom that moves outside of the box. This function is passed into ode45 as a parameter so that ode45 exits if an atom moves outside the periodic region. If ode45 exits, code execution returns to the main routine. Subsequent calculations then determine which atom crossed which boundary and return that atom to the periodic region. Then a call to timestep.m is executed to determine the next discrete time step to be passed into ode45. The current system is also passed to ode45 as the new initial vector. This process repeats each time an atom moves beyond the periodic region.

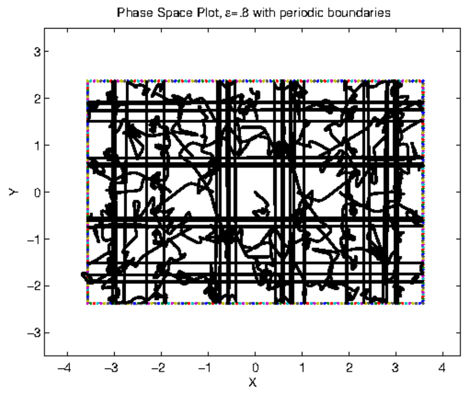
Again, the 5 by 5 system of atoms with initial uniform spacing of $.8r_{eq}$ is considered to demonstrate the effect of the periodic boundaries on the atomic lattice. As shown previously, atoms in this system are immediately repulsed due to their proximity. The periodic boundaries are now imposed so that the atoms will be confined within a periodic region. This lattice construction with periodic boundaries is simulated with a final time of $t_f = 1000$ seconds. The initial positions plot, final positions plot, and phase space plot are provided in Figure 3.12.



(a)



(b)



(c)

Figure 3.12: (a) Initial positions, (b) final positions, and (c) phase space plot for a lattice with initial uniform spacing $.8r_{eq}$ and periodic boundaries

As shown in Figure 3.12, the atoms are confined to the periodic box without the presence of rigid boundaries, and eventually equilibrate into a uniform lattice. For this boundary condition, the equilibrium configuration has uniform spacing throughout the lattice, independent of the positions of physical boundaries. This is often useful for modeling actual physical systems, making periodic boundaries the most widely used boundary condition in conventional MD simulations. With the three boundary conditions— stationary atoms, 'walls', and periodic boundaries— we now explore mixed boundaries.

3.8 Mixed Boundary Conditions

Boundary conditions can also be mixed to allow for multiple conditions in a single simulation. This provides a variety of boundary condition scenarios, depending on the physical system that is simulated. To implement the mixed conditions, the left and right boundaries are assumed to be of the same type and the top and bottom boundaries are also assumed to be of the same type, but different from that of the right and left boundaries. For example, an atomic lattice can have periodic vertical boundaries and horizontal walls. It is also possible to have no boundary conditions in one direction and one of the three boundary conditions in the other direction. Combining the four cases— no boundaries, stationary boundary atoms, walls, and periodic boundaries— gives sixteen possible scenarios that may be used as boundary conditions in MD simulation. These scenarios, along with their corresponding ode functions, are provided in Table 3.2.

Ultimately, the correct boundary condition scenario must be selected to match the physical system that is modeled. To implement any of the sixteen scenarios, the MATLAB ode45 solver is utilized by passing in the ode functions corresponding to the desired scenario. The remaining code executes according to the recipes discussed earlier in this chapter.

Table 3.2: Boundary conditions and ode functions for each mixed boundary condition scenario

Case Number	X Boundary Condition	Y Boundary Condition	LJODE Function
1	None	None	LJODE_NoBC.m
2	None	Fixed	LJODE_Fixed.m
3	None	Walls	LJODE_Walls.m
4	None	Periodic	LJODE_Periodic.m
5	Fixed	None	LJODE_Fixed.m
6	Fixed	Fixed	LJODE_Fixed.m
7	Fixed	Walls	LJODE_FixedWalls.m
8	Fixed	Periodic	LJODE_FixedPeriodic.m
9	Walls	None	LJODE_Walls.m
10	Walls	Fixed	LJODE_FixedWalls.m
11	Walls	Walls	LJODE_Walls.m
12	Walls	Periodic	LJODE_WallsPeriodic.m
13	Periodic	None	LJODE_Periodic.m
14	Periodic	Fixed	LJODE_FixedPeriodic.m
15	Periodic	Walls	LJODE_WallsPeriodic.m
16	Periodic	Periodic	LJODE_Periodic.m

3.9 Summary of Results

From physical considerations, a system of n two-dimensional second order differential equations is constructed, according to Newton's second law, and used to model the behavior of atoms over time. Systems of atoms interacting in two dimensions via Lennard-Jones forces are investigated for several types of boundary conditions. By imposing different boundary conditions on different boundaries allows for a number of mixed boundary conditions scenarios. Any of these scenarios may be used in simulation, depending on the physical system to be modeled.

Although in this thesis we have successfully implemented a two-dimensional MD simulation based on the Lennard-Jones potential, the Lennard-Jones potential function can be easily replaced by another potential, such as a REBO potential, in order to model different types of atomic lattices. Next, our MD code is utilized to model lattice defects within a Lennard-Jones model. Following the study of lattice defects within the Lennard-Jones model, empirical potentials are investigated as a next step to model lattice defects within a single layer graphene model. We conclude with suggestions for further research.

CHAPTER IV

MODELING DEFECTS OF LENNARD-JONES LATTICE

To model atomic lattice defects, the two-dimensional MD simulation implemented in Chapter 3 is used to describe the evolution of imperfect Lennard-Jones lattices over time. Here we study three incomplete bonding defects: vacancies, dislocations, and interstitials. Observing the positions of atoms within the lattice over time, in the presence of defects, provides insight into the influence of defects upon the LJ-type lattice. Furthermore it allows us to verify our MD simulations by matching the results with expected physical behavior.

In order to model defects of an LJ-type lattice, an equilibrated fully periodic lattice must first be constructed. The known equilibrium configuration that results from LJ-type atomic interactions is a triangular lattice in which every grid-point contains an atom. Adjacent atoms throughout the lattice are separated by the equilibrium distance, which, as derived previously, is $r_{eq} = 2^{1/6}$ Å. As explained earlier, r_{eq} varies slightly depending on the size of the lattice. This equilibrium configuration is shown in Figure 4.1.

Here we construct an equilibrated LJ-type lattice of $n = 480$ atoms. Other parameters for this lattice are also provided in Table 4.1. We use periodic boundaries in the x-direction as if the lattice is wrapped into a tube. In the y-direction, the

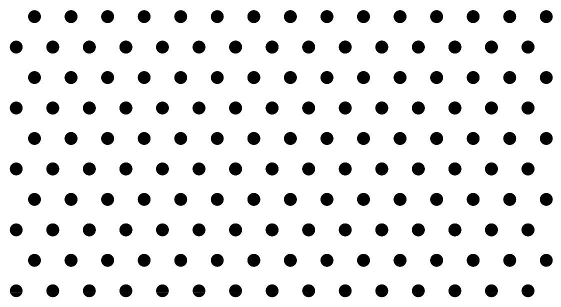


Figure 4.1: Diagram of an equilibrated lattice of atoms interacting via Lennard-Jones potential

boundaries remain free. This LJ-type lattice is used for all of the defect studies below. Since defects make an atomic lattice imperfect, once perturbed, this equilibrium configuration relaxes to a different deformed lattice for different defects. By computing the positions of atoms over time, we are able to observe the resulting lattice deformations.

Table 4.1: MD parameters used in defect study simulations

Parameter	Description	Variable
n	Number of atoms	480
m	Atomic mass	1.0 amu
σ	Separation at zero potential	1.0 Å
ϵ	Well Depth, (strength of interaction)	1.0 kJ/mol
β	Damping coefficient, (strength of damping)	1.0 eV(ps/Å)
ε	Uniform Perturbation	1
r_c	Cutoff Radius	5 Å

The code execution follows the same general algorithm for each defect study. The main file is TwoD_MDSystem.m, in which all of the parameters given in Table 4.1 are initialized, along with other MD parameters. Then the LJ-type lattice is constructed by calling the ConstructLattice.m function, which develops an equilibrated Lennard-Jones lattice with the prescribed number of rows, r, and columns, c. The function also perturbs the positions of atoms in the lattice, either uniformly by varying the parameter ε, or non-uniformly by perturbing each atom away from its equilibrium position by a small amount. Then, depending on the prescribed defect, a new function is called to create the defect in the lattice. For vacancies, dislocations, and interstitials, these functions are given in CreateVacancy.m, CreateDislocation.m, and CreateInterstitial.m, respectively. After the defect is introduced, the simulation switches to the built-in MATLAB ode45 function, which numerically solves the governing system of equations. The appropriate ode function is used depending on the type of boundary conditions, as given in Table 3.2. For this study, since we use periodic boundaries in x and free boundaries in y, LJODE_Periodic.m is the appropriate function. The remaining code execution follows the procedures discussed in Chapter 3. The only new functions in this study are those used for defect creation, which are discussed later.

4.1 Vacancies

The first type of defect studied in this thesis is a vacancy, which corresponds to a missing random lattice grid point. This creates a gap in the lattice, making it imperfect. Such a defect has been observed experimentally in graphene. Furthermore, experimental evidence shows that vacancy defects appear as a result of electron irridations, which cause both structural and mechanical changes within CNTs. Hashimoto et al., [2], use high resolution transmission electron microscopy (TEM) to observe single and multi-vacancy defects in a graphene layer, concluding that these topological defects are both numerous and stable under electron irridation. Many studies using various mathematical techniques have also been conducted to characterize and quantify the behavior resulting from vacancies. In [18], Do et al. use a nonequilibrium Green's function theory model to investigate the transport properties on graphene structures, finding that vacancies severely degrade particle transport. Lee et al., [20]-[21], utilize tight-binding molecular dynamics to simulate the diffusion of vacancy defects throughout graphene structures. They observe the evolution of the graphene lattice with both single and double vacancies, characterizing various diffusion mechanisms that occur throughout the simulations. In [22], Vozmediano et al. use a continuum model to show that vacancies in a graphene structure result in localized states at the Fermi energy, or the state of the structure at its highest energy level. In [17], Ansari et al. use MD simulation, governed by the Tersoff-Brenner REBO potential, to show that the Young's modulus and the intrinsic strength of single layered graphene sheets are both decreased by vacancies. They also suggest that smaller separation between double vacancies results in a larger reduction of strength.

In this thesis, vacancies are simulated via MD in an LJ-type lattice to observe the structural changes to the equilibrium configuration resulting from the imperfections. Vacancies have been shown to appear in an atomic lattices as both single and multi-vacancies. In multi-vacancy situations, single vacancies aggregate into one large hole in the lattice. Figure 4.2 shows the Lennard-Jones lattice with vacancies.

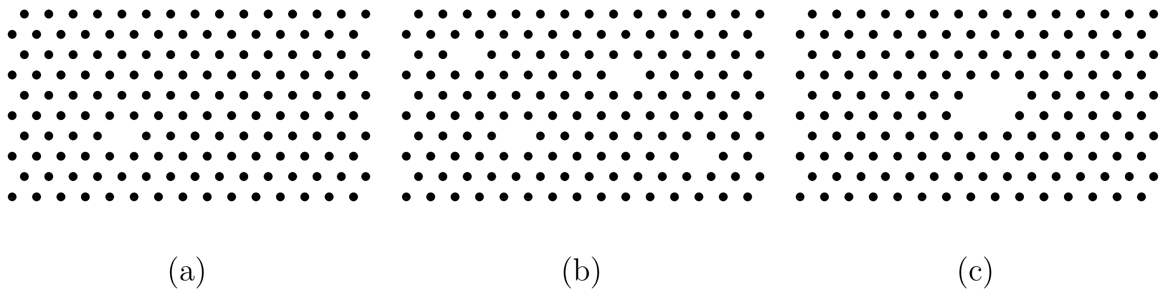


Figure 4.2: Diagrams of (a) a single vacancy, (b) multiple single vacancies, and (c) a multi-vacancy in the Lennard-Jones lattice

Vacancies are constructed using the CreateVacancy.m function, which receives the x- and y-coordinates of the LJ-type equilibrated lattice constructed in ConstructLattice.m. A vacancy is then created by either manually or randomly selecting the index of the atom to be removed. Given the index, the code then removes the x and y coordinates of the atom from the positions vector. This is done any number of times, depending on the desired number of vacancies in the system. The positions vector and the updated number of atoms are returned to the main file and the simulation begins by calling the ode45 solver, as explained previously.

In the first simulation a single vacancy is generated by creating a gap at the center of the $n = 480$ LJ-type lattice, and is simulated via MD. The positions of the atoms in the lattice are then graphed over time in order to observe its evolution.

In Figure 4.3, the voronoi diagrams for the lattice of atoms are shown at simulation times $t = 0$, $t = 3$, $t = 5$, and $t = 200$ seconds.

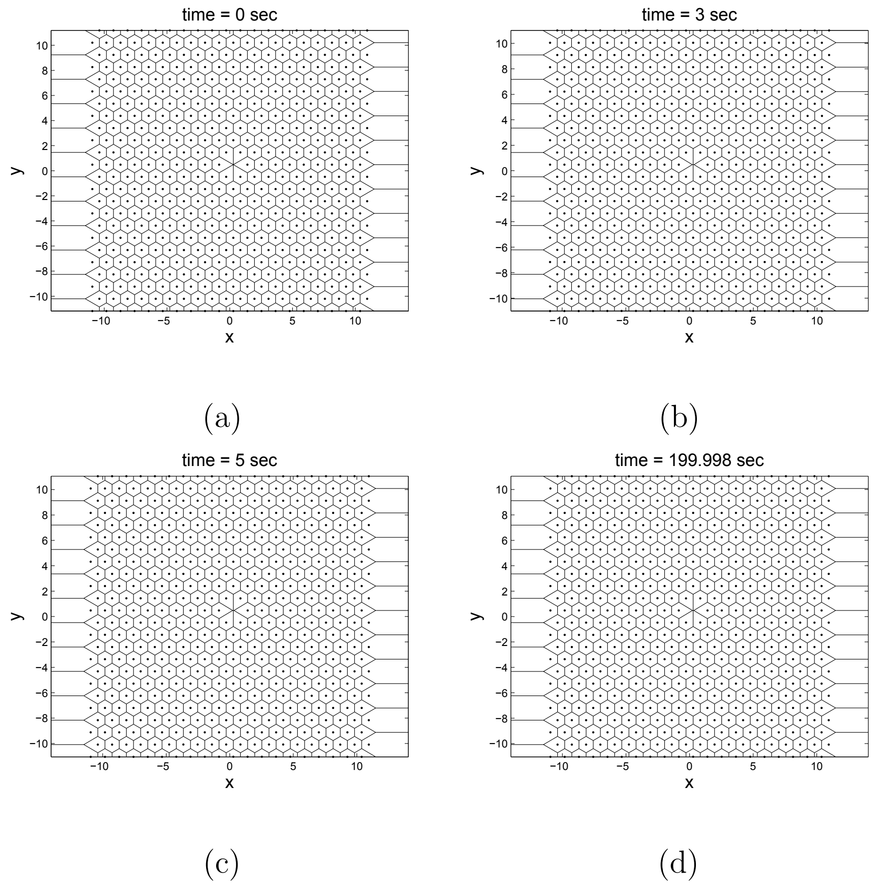


Figure 4.3: Positions of atoms in an LJ-type lattice with a single vacancy after (a) 0 seconds, (b) 3 seconds, (c) 5 seconds, (d) 200 seconds.

The first graph, at $t = 0$, illustrates the initial positions of the lattice, showing the vacancy defect near the center of the lattice. After 3 seconds, the atoms surrounding the vacancy move slightly towards their neighbors and the remaining atoms shift slightly inward. Note that this is because the atoms are initially separated by $r_{eq} = 2^{(1/6)}$ Å, while the equilibrium spacing is actually slightly less. After 5 seconds, the vacancy still appears at the center. We graph the system at 200 seconds to show that after a long runtime, the defect remains inside the lattice. It is likely that the energy in the system is higher due to the vacancy, but this energy is not large enough for the vacancy to diffuse to the boundary. It is also possible that given a longer time scale, there would be additional movement in the lattice, since very small movements over time can eventually become large movements. However, on the time scale of our study, the vacancy remains in the center of the lattice and does not diffuse to the boundary.

Next we simulate a multi-vacancy by creating a larger gap in the center of the system. Effects upon the LJ-type lattice are more likely to occur in this setup, since the larger gap should result in larger energies at the center of the lattice. Figure 4.4 graphs the positions of the atoms at simulation times $t = 0$, $t = 3$, $t = 5$, and $t = 200$ seconds. In the figure, it is shown that similar behavior to that of the single vacancy occurs— the atoms collapse slightly to settle at their equilibrium distances, but over time the large gap at the center remains. Again, it is possible that more lattice deformation could occur on a large time scale.

We finish this study of vacancy defects by constructing multiple single vacancies in the lattice and investigating the resulting behavior of the lattice. To create multiple single vacancies, five atoms separated by variable distances are removed from the lattice. Since the energy in the previous multi-vacancy model was not high enough to diffuse the defect to the boundary, it is not expected for the multiple vacancy sites to diffuse. As in the previous vacancy simulations, the positions of atoms are graphed

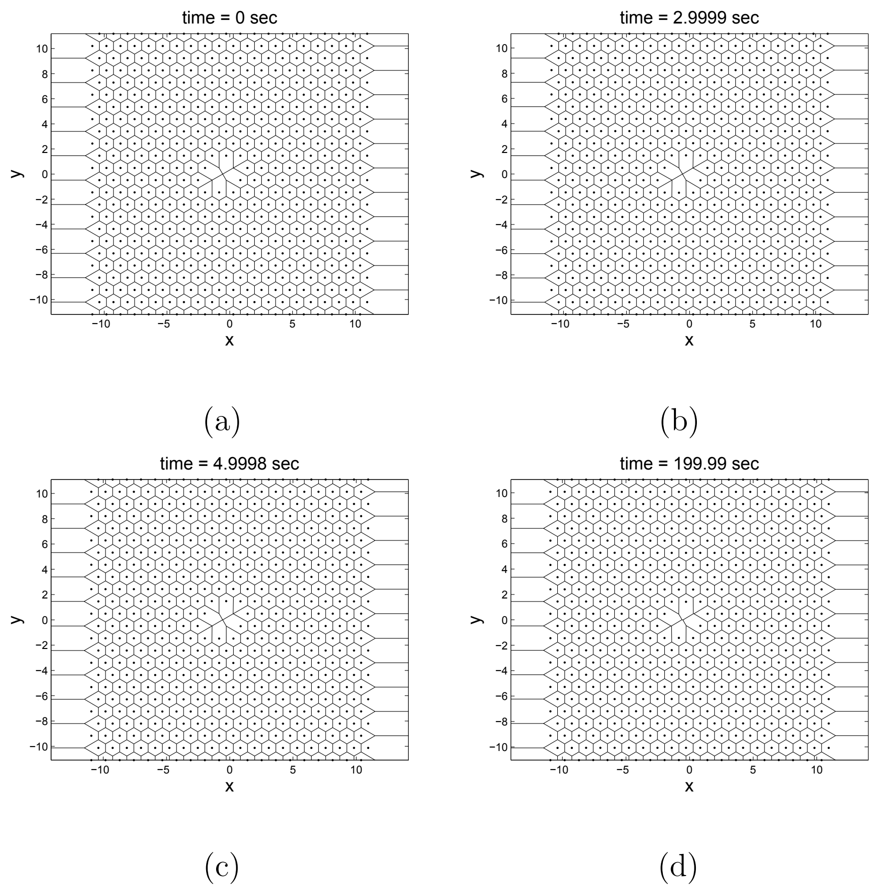
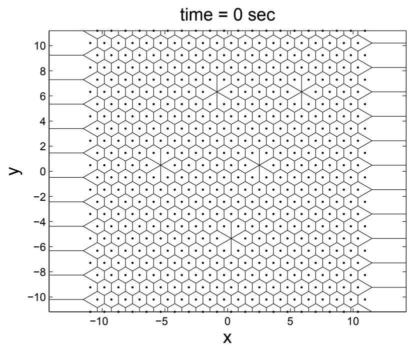


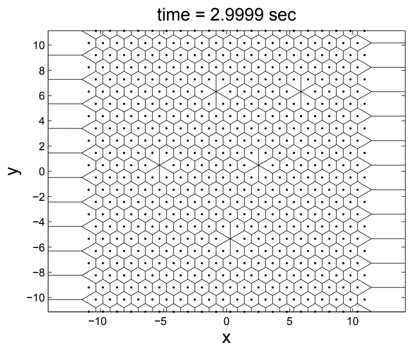
Figure 4.4: Positions of atoms in an LJ-type lattice with a multi-vacancy after (a) 0 seconds, (b) 3 seconds, (c) 5 seconds, (d) 200 seconds.

at $t = 0$, $t = 3$, $t = 5$, and $t = 200$, to show the evolution of the imperfect lattice, and provided in Figure 4.5.

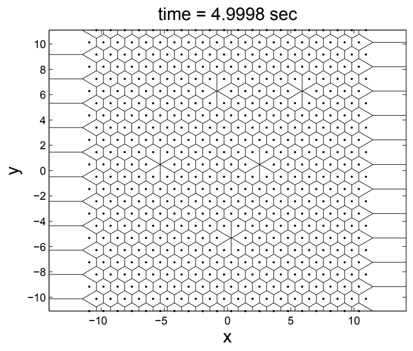
As expected, little to no movement is observed in the lattice over time. As in the previous cases, atoms surrounding vacancies move outward slightly while the remaining atoms collapse inward very slightly, while settling into their equilibrium positions. The graph of positions at $t = 200$ seconds shows that over time, the vacancies do not diffuse from the LJ lattice. This could suggest that vacancies are, perhaps,



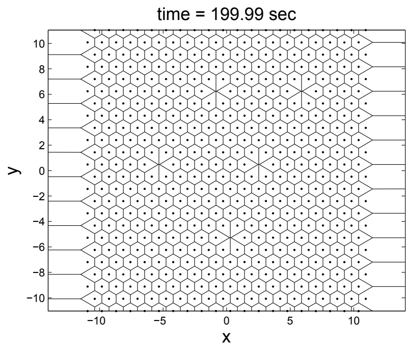
(a)



(b)



(c)



(d)

Figure 4.5: Positions of atoms in an LJ-type lattice with multiple single vacancies after (a) 0 seconds, (b) 3 seconds, (c) 5 seconds, (d) 200 seconds.

stable in the LJ-type lattice. It is also possible that if a high enough percentage of atoms are removed, creating many vacancies in the lattice, that more movement in the system would be observed. These conjectures can only be assumed for the LJ-type lattice. It is, on the other hand, expected that vacancies would have larger impact upon a graphene, since it creates incomplete bonds within the graphene layer.

4.2 Dislocations

The second defect considered in this thesis is a dislocation. Dislocations are irregular formations of atoms associated with insertion or removal of partial rows from the lattice. Several types of dislocation defects have been observed in graphene via transmission electron microscopy (TEM). As suggested by Ariza and Ortiz in [23], dislocations lead to deleterious effects for example on mechanics of the lattice, making the understanding of mechanical properties of graphene even more important, so that the results upon equilibrium and kinetic properties of graphene can be characterized. Many studies have been conducted with this aim. Carpio et al., [1], use tight binding calculations to study glide and shuffle dislocations in graphene, which result from irridation of graphene sheets. Their aim is to show that such dislocations are stable generate magnetic moments in graphene sheets, giving rise to unique magnetic properties. Juan et al., [24], utilize a continuum model to observe effects of dislocations upon the electronic properties in a graphene sheet, finding that dislocations in graphene couple in the form of vector gauge fields. In [25], Yazyev et al. use Burgers vectors to construct dislocations in graphene, and first principle calculations are performed using density-functional theory. From this study it is suggested, in general, that this defect has strong effects on the electronic structure of a graphene layer.

In our study, a dislocation is constructed in the Lennard-Jones lattice, and the effect upon the equilibrium configuration is observed. A dislocation can be created by inserting an extra row of atoms into the lattice, as in Figure 4.6.

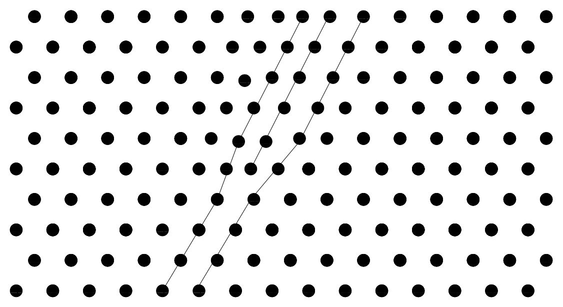
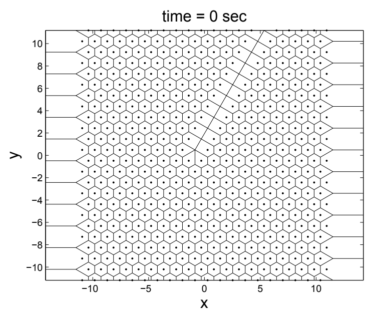


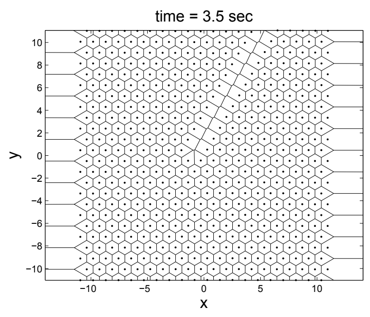
Figure 4.6: Diagram of a dislocation in the Lenard-Jones lattice

A dislocation in the LJ-type lattice is created by calling the CreateDislocation.m function. This function receives the x- and y-coordinates of atoms forming the LJ-type lattice, then uses these coordinates to identify the top half of the atoms in a given diagonal row. This row is removed from the lattice in order to create the dislocation. The row from which the atoms are removed can be selected either manually or randomly. After identifying the indices of the atoms to be removed, their positions are deleted from the positions vector, and the total number of atoms in the system is updated. This information is then returned to the main routine and the lattice is then simulated via MD.

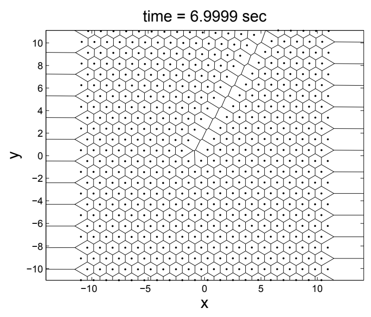
Using this setup, a dislocation is created in the LJ-type lattice with 480 atoms and simulated with sufficient time for the system to equilibrate. The results from this simulation are given in Figure 4.7. In the figure, the positions of atoms in the system are graphed at simulation times $t = 0$, $t = 3.5$, $t = 7$, $t = 9.5$, $t = 11.5$, $t = 13.5$, $t = 14.5$, $t = 16.5$, and $t = 26$ seconds.



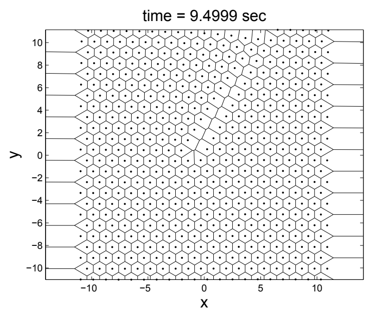
(a)



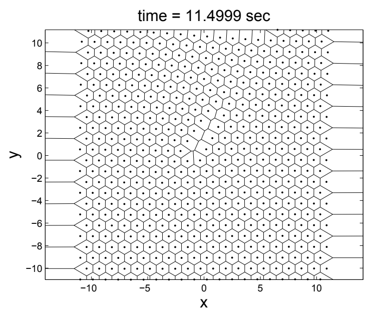
(b)



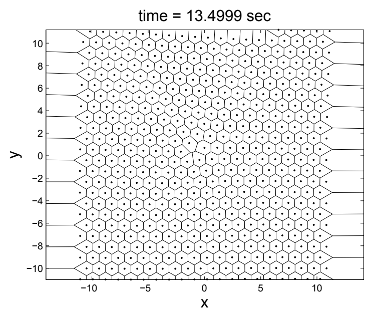
(c)



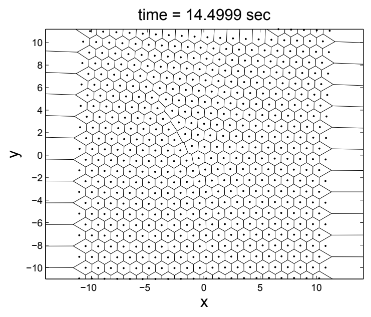
(d)



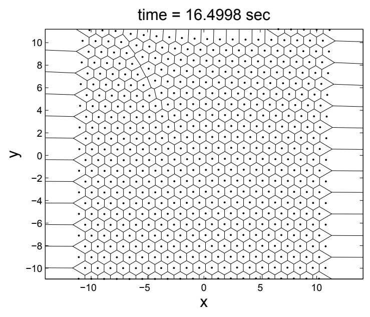
(e)



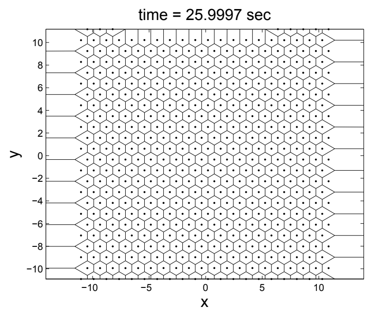
(f)



(g)



(h)



(i)

Figure 4.7: Positions of atoms in an LJ-type lattice with a dislocation after (a) 0 seconds, (b) 3.5 seconds, (c) 7 seconds, (d) 9.5 seconds, (e) 11.5 seconds, (f) 13.5 seconds, (g) 14.5 seconds, (h) 16.5 seconds, and (i) 26 seconds.

This defect simulation shows a significant movement of atoms near the defect, as the lattice initially deforms due to the missing atoms. The graph at $t = 0$ shows the lattice in its initial state, in which the cut is introduced from near the center of the system diagonally upward to the surface. At $t = 3.5$ seconds, movement is already observed where the atoms at the edges of the cut move towards the gap. This movement is even more evident at $t = 7$ seconds. After 9.5 seconds, the atoms left of the cut are attracted further towards the atoms on its right side, as the gap is "zipped" toward the center of the lattice. After 13.5 seconds, the defective region has completely concentrated near the lower end of the cut; the area where the rest of the defect once existed appears to be close to the equilibrium configuration. The energy in the system is still large enough that the newly formed dislocation begins to diffuse towards the surface, which is observed at $t = 14.5$ seconds, and moves even closer towards the surface after 16.5 seconds. After 26 seconds, the dislocation has annihilated at the surface, leaving a gap in the top row of atoms. Elsewhere, the entire lattice has reached an equilibrium configuration that is similar to that of a perfect LJ-type lattice.

4.3 Interstitial Atoms

The final defect studied in this thesis is the interstitial defect, in which in an extra atom is inserted into the atomic lattice. A grid-point for this atom does not exist in the lattice, so the spacing between the interstitial and its nearest neighbors is less than r_{eq}, resulting in large repulsive forces. This is a known defect in many carbon lattices, including graphene and other crystalline structures, which may appear following irridation of carbon lattices. A number of studies have been conducted to model interstitials in graphene, among other materials. Orlita et al. [26], show the existence of the interstitial defect via magneto-spectroscopy. Using this experimental technique, they observe interstitials in a graphene bilayer, suggesting that it leads to the departure of Landau levels in graphene, and hence limiting electronic bands. Shodja et al., [27], use both continuum theory and MD simulation to model the behavior of fcc films with interstitial atoms. Their focus is upon the effect of the extra atom, dependent upon its proximity to the surface of the film. Using an empirical potential and the Velocity-Verlet algorithm, their MD results closely match those from the continuum model. A computational study is also conducted by Gulans et al., [19], in which density-functional theory is used to study the energetics of single interstitials and di-interstitials in a graphene bilayer. Their results support that spiro interstitial defects are the most stable within graphene, while dumbbell interstitials are the least stable. It is also supported that di-interstitials result in more lattice deformations than single interstitials.

In this study, interstitials are introduced into the Lennard-Jones lattice in order to observe their influence on the equilibrium configuration. We simulate the evolution of the LJ-type lattice of 480 atoms with both single and double interstitials, with the conjecture that a double interstitial should result in more lattice deformation. Indeed, it is likely that more interstitials will lead to higher energy in the system, resulting in more lattice deformation. Figure 4.8 shows a schematic of both single and double interstitial defects in the LJ-type lattice. The region of the defects are circled in the figures.

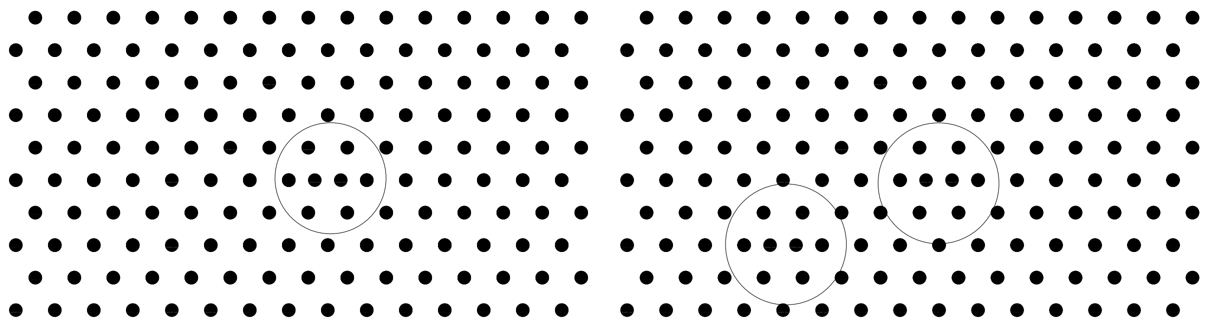
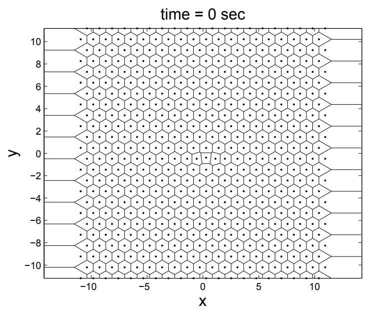


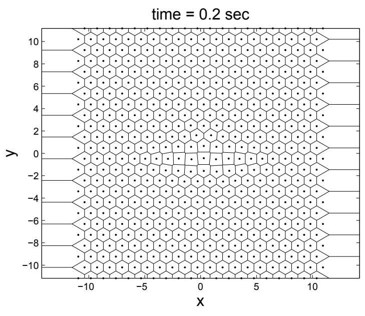
Figure 4.8: Diagrams of (a) a single interstitial and (b) a double interstitial within the Lenard-Jones lattice.

Interstitials are created by implementing the CreateInterstitial.m function, which receives the coordinates of the atoms in the LJ-type lattice as an input. Given these coordinates, the indices of the two atoms that the interstitial should be placed between are chosen, and the interstitial is inserted between the two atoms. The coordinates of the extra atom are then added to the positions vector, and the number of atoms is updated. This can be done for any number of interstitials. The function then passes the new n value and positions vector back to the main routine to execute the MD simulation.

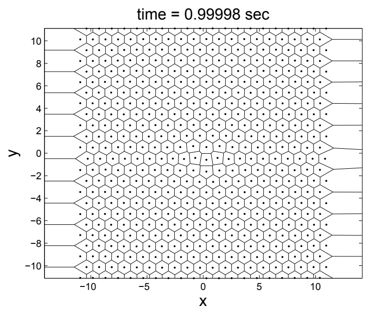
First, a single interstitial is inserted into the LJ-type lattice by shifting an atom over, and inserting an extra atom in an equilibrated region. The resulting lattice deformation is observed. For this defect, we observe the evolution of the imperfect lattice by graphing the positions of atoms at $t = 0$, 0.2, 1, 1.5, 3, 50, 100, 150, and 200 seconds. The graphs are provided in Figure 4.9.



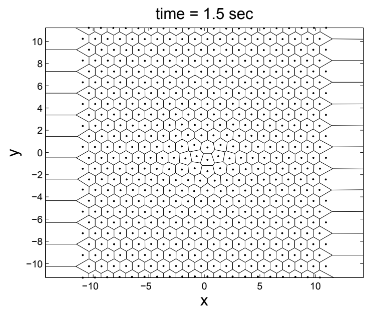
(a)



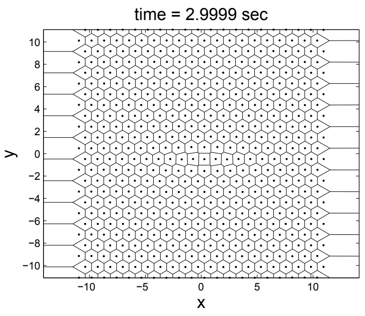
(b)



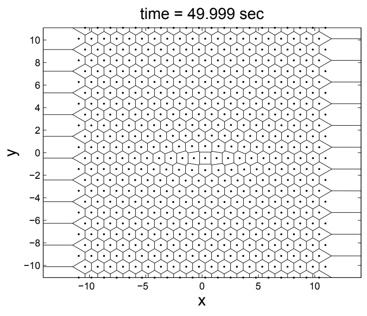
(c)



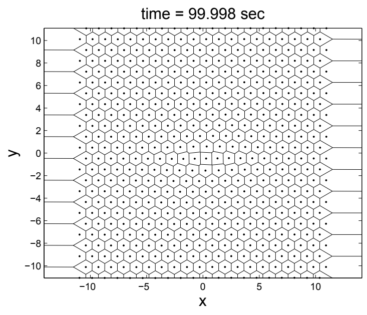
(d)



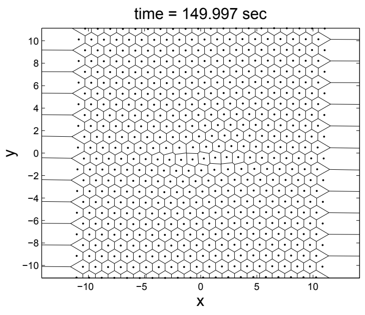
(e)



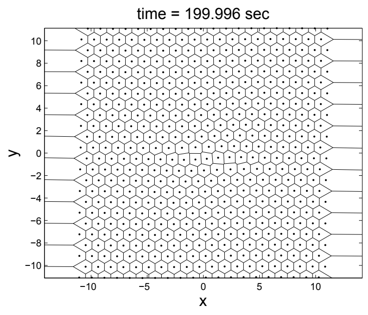
(f)



(g)



(h)



(i)

Figure 4.9: Positions of atoms in an LJ-type lattice with a single interstitial after (a) 0 seconds, (b) 0.2 seconds, (c) 1 second, (d) 1.5 seconds, (e) 3 seconds, (f) 50 seconds, (g) 100 seconds, (h) 150 seconds, and (i) 200 seconds.

It is shown in the figure that due to the interstitial defect, initially there are large repulsive forces, so that after only .2 seconds repulsion is observed in the center of the lattice near the defect site. The repulsion continues outward towards the edges of the lattice, and after 1 second, appears to dissipate. After 3 seconds the atoms appear to have equilibrated, though there is still some deformation at the center of the lattice. By 50 seconds, the lattice has experienced little to no additional movement and the deformation still exists. However, after about 100 seconds of simulation time, the lattice shows more movement, as the defective region now appears to diffuse towards the boundary. After 200 seconds the movement ceases and the lattice appears to have reached a stable, equilibrated state. Though the lattice is still deformed, the deformation is much less pronounced. It is still possible that, on a much larger time scale, further movement could entirely diffuse the defect to the boundary of the lattice.

In the final simulation, a double interstitial defect is created in the LJ-type lattice by placing two extra atoms in the system, separated by some distance. To observe the impact of two interstitials upon the lattice, the positions of atoms are given at $t = 0$, 0.1, 0.2, 1, 1.5, 3, 50, 100, and 200 seconds. The graphs for this double interstitial case are provided in Figure 4.10.

As in the previous simulation, initial repulsion occurs due to the presence of extra atoms in the system. As shown in the figure, large repulsive forces can be observed after only 0.1 seconds at both interstitial sites. After only 0.2 seconds this repulsion becomes even larger, and by 1 second, the deformation wave appears to have reached to the boundary of the lattice. However, there still remains a large amount of movement throughout the system, especially near the defect sites as the

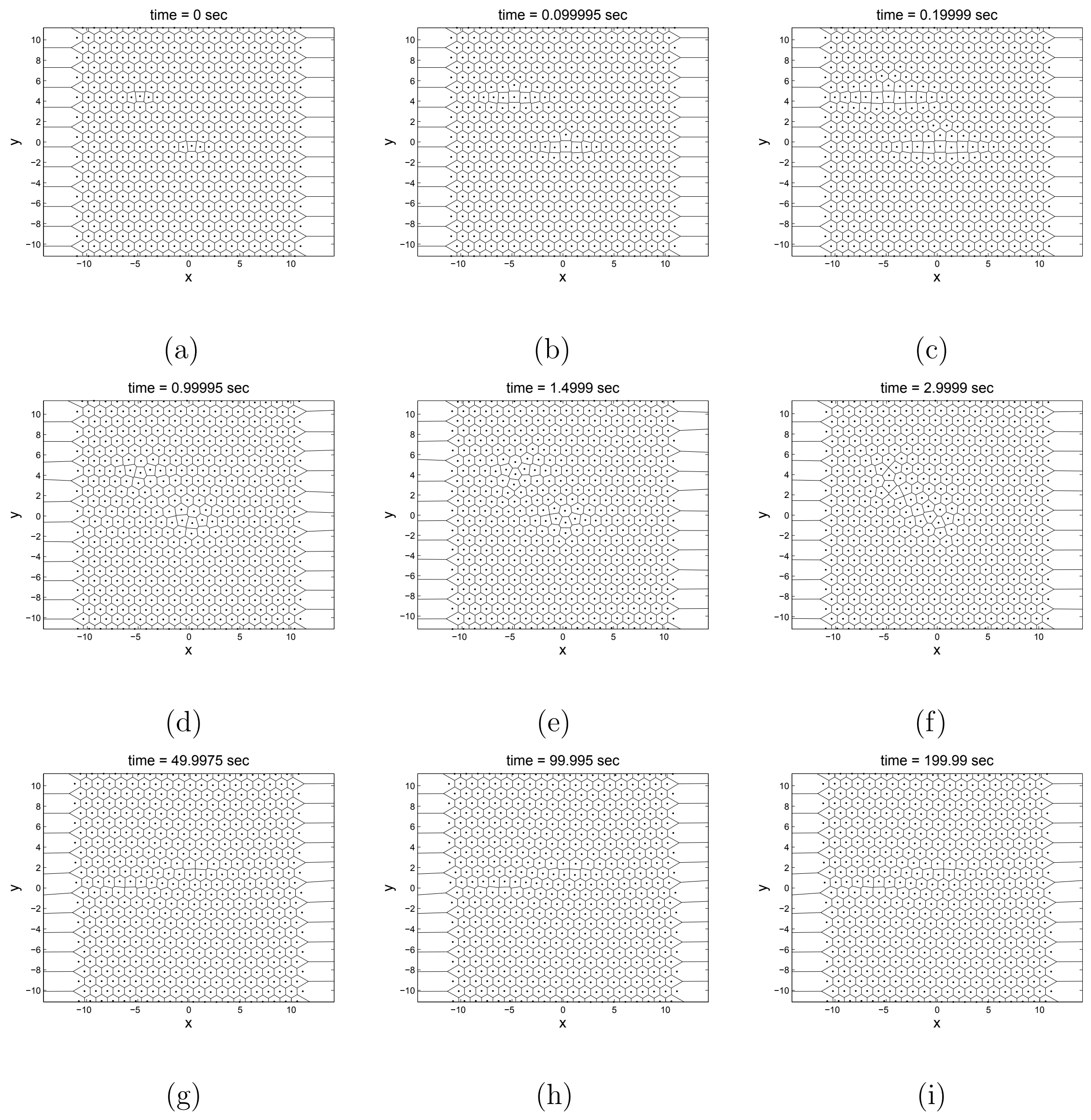


Figure 4.10: Positions of atoms in an LJ-type lattice with a double interstitial after (a) 0 seconds, (b) 0.1 seconds, (c) 0.2 seconds, (d) 1 second, (e) 1.5 seconds, (f) 3 seconds, (g) 50 seconds, (h) 100 seconds, and (i) 200 seconds.

interstitials move closer to each other. After around 50 seconds, the lattice appears to have equilibrated and the graphs at 100 and 200 seconds show that there is no additional movement in the system. While there remains deformation in the lattice as a result of the two interstitials, areas of the lattice further from the defect sites are close to the LJ-type equilibrium configuration. We also note that, in support of our conjecture, double interstitials appear to result in more lattice deformation than the single interstitial atom.

4.4 Summary of Results

By implementing the MD simulation developed in Chapter 3, we simulate the behavior of defects in an LJ-type lattice over time. This is achieved for three known defects: vacancies, dislocations, and interstitials. Our results match the expected behaviors resulting from these defects. First, we model vacancies in the lattice by simulating a single vacancy, multi-vacancy, and multiple single vacancies. Each of the simulations show the same behavior surrounding the defect sites, and in every case, the defect remains in the lattice at its initial site. We conclude in this study that perhaps, on a larger time scale, move movement would occur in the system. However, on our time scale, the initial energies are not large enough to diffuse vacancies to the boundaries. In graphene this defect should result in more noticeable behavior, since it would break atomic bonds at the vacancy sites. Next, we simulate a dislocation in the lattice. The resulting behavior over time shows that the dislocation leads to bending in the lattice. This bending first diffuses towards the center of the lattice where the defect begins, then diffuses back towards the boundary. In this simulation, the defect is completely diffused from the lattice, leaving a perfect LJ-type equilibrium configuration. Finally, we simulate interstitials, both single and double, which shows that in both cases the resulting lattice deformations are not removed from the lattice. As in the vacancy study, it is possible that further diffusion of the interstitials would occur on larger time scale. Furthermore, we show that more deformation occurs in the double interstitial simulation.

These results are consistent with the expected behaviors resulting from defects in the LJ-type lattice. The next steps should be to model these defects in graphene. While the defects are the same, we expect slightly different behavior since the equilibrium configuration for graphene is different than that for a Lennard-Jones lattice. Also, the Lennard-Jones potential is a non-bonded potential, while the REBO potential— which should be used to represent atomic interactions in a graphene layer— is a bonded potential, modeling the covalent bonds of carbon atoms. In the next section we discuss this REBO potential, and conclude with suggestions for further research.

CHAPTER V

MODELING GRAPHENE LATTICE USING REBO POTENTIAL

5.1 Reactive Empirical Bond Order Potentials

To accurately model the behavior of a graphene sheet, a potential energy function that represents the behavior of carbon atoms must be used. The Lennard-Jones potential is a long-range, non-bonded interatomic pair potential that is useful in modeling more basic systems with non-bonded atoms. Since carbon atoms are covalently bonded atoms, the Lennard-Jones potential does not accurately represent the behavior of a graphene sheet. However, because carbon atoms have uniquely strong bonds and a high melting point, unlike other known elements, it has become very important to develop a new interatomic potential for carbon [28]. This potential is called a reactive empirical bond order potential.

The main purpose of a potential function is to accurately and effectively model specific atomic systems. It must also be flexible so that it can fit a variety of lattice sizes and structures while accurately modeling those lattices. Potential energy functions must also be computationally efficient, since they are evaluated over several time steps through MD simulation. Empirical interatomic potentials are analytic potential energy functions that fit this criterion, which are used to model the behavior and structural properties of more specific complex atomic systems. The reactive empirical bond order (REBO) potential is used to model the behavior of carbon and hydrocarbon atoms, such as those in a graphene sheet. Unlike the Lennard-Jones potential, which is a function of the distance between atoms, the REBO potential is dependent upon both the distance and the vector angle between atoms. It also takes into account their surrounding local environment. Since a single layer graphene sheet is a monolayer of covalently bonded carbon atoms, the REBO potential can be used in MD simulation to model its behavior.

Many studies have been conducted prior to the development of the now widely used second generation REBO potential by Brenner. The first empirical potential was introduced by Abell in [29], where the local environment surrounding atoms is found to have an effect upon their binding energies. Using chemical pseudo-potential theory, it is determined that atomic binding energies are a function of separate attractive and repulsive terms, giving the general formalism,

$$E_B = \sum_k Z_k (q_k V_{R_k} + p_k V_{A_k}), \tag{5.1}$$

where E_B is the binding energy per atom, and V_R and V_A are repulsive and attractive terms such that

$$V_R = Ae^{(-\lambda_1 r)} \tag{5.2}$$

and

$$V_A = -Be^{(-\lambda_2 r)}. \tag{5.3}$$

In the equations, A, B, λ_1, and λ_2 vary according to the atomic system, where A and B are the strengths of attraction and repulsion, and λ_1 and λ_2 are constants representing the speed at which these strengths decay. The distance between atoms is given by r. It is suggested that the repulsive and attractive terms should be exponential equations because atomic energies decay exponentially as the distance between atoms increases [29].

Following the works of Abell, Tersoff expands on the model in (5.1) to develop an empirical energy formalism for covalent systems. Here, the strength of each atomic bond, or bond order, depends on the local environment so that atoms with more neighbors form weaker bonds than atoms with fewer neighbors. Under the assumption that the bond order depends on both the atomic geometry and surrounding atomic neighbors in a system, an empirical potential for silicon, a covalent element, is developed. In this model, the interatomic potential has the form

$$E_b = \frac{1}{2}\sum_{i\neq j} f_c(r)[a_{ij}f_R(r) + b_{ij}f_A(r)], \tag{5.4}$$

where f_R and f_A are equivalent to V_R and V_A, respectively, from the Abell model, and $f_c(r)$ is a cutoff function such that

$$f_c = \begin{cases} 1, & r < R - D \\ \frac{1}{2} - \frac{1}{2}\sin(\frac{\frac{\pi}{2}(r-R)}{D}), & R - D \\ 0, & r > R + D \end{cases} \tag{5.5}$$

The parameters a_{ij} and b_{ij} have functional forms depending on the position of atoms, so that the resulting interatomic potential is dependent upon both the bond distance and bond angle. Based on the Tersoff model, the interatomic potential is a sum over an atom's nearest neighbors [30]. Tersoff builds upon this model in following works, developing an empirical interatomic potential specifically for carbon [28]. In this study, a set of discrete parameter values are given, which are chosen by fitting energies of carbon prototypes, and used with the original formalism to model carbon atomic bonding. Using this model with Monte Carlo and periodic boundaries, a carbon lattice is simulated to display the effects of atomic defects. These results are compared to experimental data, and the empirical potential is found to accurately describe the energies of carbon [28].

The Tersoff model is unable to reproduce many properties of carbon, such as defining the difference between conjugated or non-conjugated double bonds. This suggests that the interatomic potential should be a sum over bonds rather than over nearest neighbors, such that

$$E_B = \sum_i \sum_{j(>i)} [V_R(r_{ij}) - \bar{b}_{ij} V_A(r_{ij})], \tag{5.6}$$

where V_R and V_A are repulsive and attractive terms, and $\bar{b}_{ij}$ has the form

$$\bar{b}_{ij} = (B_{ij} + B_{ji})/2. \tag{5.7}$$

The terms $\bar{b}_{ij}$ and $\bar{b}_{ji}$ have functional form to represent the bonds between pairs of atoms, accounting for conjugated or non-conjugated bonding. This form calculates binding energies in a carbon lattice of 7.3768 eV per atom, with atoms separated by 1.42 Å, which is the distance between bonded carbon atoms in a graphene lattice [31].

5.2 Second-Generation REBO Potential

Brenner et al. later improve the Brenner model, developing the second generation reactive empirical bond order potential [32]. The second-generation REBO potential allows for covalent bond breaking and forming with associated changes in an atomic environment, making it an ideal interatomic potential to model defects in carbon atomic lattices such as graphene. The second-generation potential is a revised model from Brenner's original work, in which improved analytic functions are given, along with an extended fitting database for more accurate calculations of bond lengths, energies, and force constants [32]. The bond energy, E_b, is written as a sum over both bonds and nearest neighbors, such that

$$E_b = \sum_i \sum_{j(>i)} [V_R(r_{ij}) - \bar{b}_{ij} V_A(r_{ij})], \tag{5.8}$$

where V_R and V_A are the repulsive and attractive terms, such that

$$V_R(r) = f_c(r)(1 + \frac{Q}{r})Ae^{-\alpha r}, \tag{5.9}$$

and

$$V_A(r) = f_c(r) \sum_{n=1}^{3} B_n e^{-\beta_n r}, \tag{5.10}$$

where A and B define the strengths of repulsion and attraction respectively. The parameters A and B, as well as α and β, which represent the speed of decay, vary based on the atomic system. Also, f_c is a cutoff function that limits the range of atomic interactions. The cutoff function is a piecewise function dependent upon the distance between atoms, where as in the Tersoff model,

$$f_c(r) = \begin{cases} 1, & r < R - D \\ \frac{1}{2} - \frac{1}{2}\sin(\frac{\frac{\pi}{2}(r-R)}{D}), & R - D \\ 0, & r > R + D \end{cases} \tag{5.11}$$

so that $R + D$ defines the range where the potential is cut off, and $R - D$ defines the range where the potential is not reduced. The bond order, or bond strength, is defined by

$$\bar{b}_{ij} = \frac{1}{2}(b_{ij}^{\sigma-\pi} + b_{ji}^{\sigma-\pi}) + b_{ij}^{\pi}, \tag{5.12}$$

where $b_{ij}^{\sigma-\pi}$ and $b_{ji}^{\sigma-\pi}$ depend on the bond angles and local atomic environment [33]. For a graphene lattice, $b_{ij}^{\sigma-\pi} = b_{ji}^{\sigma-\pi}$ due to symmetry, and

$$b_{ij}^{\sigma-\pi} = (1 + \sum_{k \neq i}^{j} f_{ik}^{c} g_{ijk})^{-\frac{1}{2}}. \tag{5.13}$$

The term b_{ij}^{π} defines the covalent bonding of carbon atoms and is given by the sum of two terms such that,

$$b_{ij}^{\pi} = \prod_{ij}^{RC} + b_{ij}^{DH}, \tag{5.14}$$

where $\prod_{ij}^{RC}$ and b_{ij}^{DH} are determined based on the bond between atoms i and j and the carbon-to-carbon bond angle respectively. Since $\prod_{ij}^{RC}$ accounts for atomic bonds, it also takes into account lattice defects. In an atomic system without defects, $\prod_{ij}^{RC}=0$. The second term, b_{ij}^{DH}, depends on the bond angle and represents a bending function, which is vastly important in modeling graphene and SWCNTs [33]. This is given by

$$b_{ij}^{DH} = \frac{T_0}{2} \sum_{k,l \neq i,j} f_{ik}^{c} f_{jl}^{c} (1 - \cos^2 \phi), \tag{5.15}$$

where ϕ is the dihedral angle between an atom and its three nearest neighbors. The function g_{ijk} is dependent upon the angle between atoms i, j, and k, and is given by

$$g_{ijk} = \sum_{i=0}^{5} \lambda_i \cos^i(\theta_{ijk}). \tag{5.16}$$

All constants in this second generation REBO potential vary depending on the specific atomic system and the types of bonds between atoms. Several studies have been conducted in effort to parameterize this REBO potential and find optimal values for the constants. In [32], such values are provided for both carbon and hydrogen atoms and their bonds. In [33], Lindsay et al. develop a method to determine optimized values for these constants, which are said to have results accurate to experimental data for graphene. The authors claim that the original parameter values provided by Brenner do not accurately calculate the photon dispersions in a graphene sheet. With their optimized constant values, the calculated thermal conductivity values and atomic velocities in SWCNTs are in better agreement with experimental data. In (5.1), the original Brenner values are given, as well as the optimized Lindsay values. While the numbers vary only slightly, the slight differences have a large effect on the nano scale and noticeable accuracy in modeling graphene.

5.3 Further Research

The second generation REBO potential can be used to accurately model the interaction between carbon atoms within a graphene layer. In suggestion for further research, the two-dimensional MD simulation developed in Chapter 3 and utilized

Table 5.1: Original parameter values from the Brenner model and the optimized Lindsay values

Parameter	Brenner Values	Optimized Lindsay Values
A	10953.544162170 eV	10953.544162170 eV
B_1	12388.79197798 eV	12388.79197798 eV
B_2	17.56740646509 eV	17.56740646509 eV
B_3	30.71493208065 eV	30.71493208065 eV
α	4.7465390606595 $\AA^{-1}$	4.7465390606595 $\AA^{-1}$
β_1	4.7204523127 $\AA^{-1}$	4.7204523127 $\AA^{-1}$
β_2	1.4332132499 $\AA^{-1}$	1.4332132499 $\AA^{-1}$
β_3	1.3826912506 $\AA^{-1}$	1.3826912506 $\AA^{-1}$
Q	0.3134602960833 Å	0.3134602960833 Å
R	2.0 Å	2.0 Å
D	1.7 Å	1.7 Å
T_0	-0.00809675	-0.0165
λ_0	0.7073	0.0000
λ_1	5.6774	-3.1822
λ_2	24.0970	-19.9928
λ_3	57.5918	-51.4108
λ_4	71.8829	-61.9925
λ_5	36.2789	-29.0523

in Chapter 4 should be used to model defects within a graphene layer. While the respective results are expected to be similar to those in Chapter 4, it is necessary to understand the role of defects in CNTs because these defects could influence the many properties that make them so useful.

To accomplish this, an equilibrated graphene layer should first be constructed. The three defects modeled with the Lennard-Jones model— vacancies, dislocations, and interstitials— can then be imposed upon the graphene layer. Using two-dimensional MD simulations, the effects of each defect upon the equilibrated graphene layer should then be investigated. The second generation REBO potential, coupled with the parameters given by Lindsay, et al., and provided in (5.1), should be used to model short-range bonded carbon interactions. To model long-range atomic interactions, the Lennard-Jones potential should be utilized to model the non-bonded interactions between carbon atoms in the system. By coupling the REBO potential for bonded short-range interactions with the Lennard-Jones potential for long-range, non-bonded interactions, all possible atomic interactions within the graphene layer are accounted for. Finally, full three-dimensional simulations should be conducted to account, for example the rippling of graphene layers. These suggestions for further research, should yield results to provide insight into the mechanical behavior of graphene.

BIBLIOGRAPHY

[1] A. Carpio, L. Bonilla, F. Juan, and M.A.H. Vozmediano. Dislocations in graphene. *New Journal of Physics*, 10, May 2008.

[2] A. Hashimoto, K. Suenaga, A. Gloter, K. Urita, and S. Iijima. Direct evidence for atomic defects in graphene layers. *Nature*, 430(7002):870–873, Aug 2004.

[3] M. Taghioskoui. Trends in graphene research. *Materials Today*, 12(10):34–37, Oct 2009.

[4] G. Odegard, T. Gates, L. Nicholson, and K. Wise. Equivalent-continuum modeling of nano-structured materials. *Composites Science and Technology*, 62(14):1869–1880, Nov 2002.

[5] M. Shokrieh and R. Rafiee. Prediction of Young's modulus of graphene sheets and carbon nanotubes using nanoscale continuum mechanics approach. *Materials & Design*, 31(2):790–795, Feb 2010.

[6] C. Neto, F. Guinea, N. Peres, K. Novoselov, and A. Geim. The electronic properties of graphene. *Rev. Mod. Phys.*, 81(1):109–162, Jan 2009.

[7] N. Mohanty and V. Berry. Graphene-based single-bacterium resolution biodevice and DNA transistor: interfacing graphene derivatives with nanoscale and microscale biocomponents. *Nano Letters*, 8(12):4469–4476, Dec 2008.

[8] T. Booth, P. Blake, R. Nair, D. Jiang, E. Hill, U. Bangert, A. Bleloch, M. Gass, K. Novoselov, M. Katsnelson, and A. Geim. Macroscopic graphene membranes and their extraordinary stiffness. *Nano Letters*, 8:2442–2446, Aug 2008.

[9] P. Ayajan and O. Zhou. Applications of carbon nanotubes. *Topics in Applied Physics*, 80:391–425, 2001.

[10] A. Bianco, K. Kostarelos, and M. Prato. Applications of carbon nanotubes in drug delivery. *Current Opinion in Chemical Biology*, 9(6):674 – 679, Dec 2005.

[11] S. Georgantzinos, G. Giannopoulos, and N. Anifantis. Numerical investigation of elastic mechanical properties of graphene structures. *Materials & Design*, 31(10):4646–4654, Dec 2010.

[12] A. Hemmasizadeh, M. Mahzoon, E. Hadi, and R. Khandan. A method for developing the equivalent continuum model of a single layer graphene sheet. *Thin Solid Films*, 516(21):7636–7640, Sep 2008.

[13] H. Bu, Y. Chen, M. Zou, H. Yi, K. Bi, and Z. Ni. Atomistic simulations of mechanical properties of graphene nanoribbons. *Physics Letters A*, 373(37):3359–3362, Sep 2009.

[14] Z. Ni, H. Bu, M. Zou, H. Yi, K. Bi, and Y. Chen. Anisotropic mechanical properties of graphene sheets from molecular dynamics. *Physics B-Condensed Matter*, 405:1301–1306, Mar 2010.

[15] W. Bao, C. Zhu, and W. Cui. Simulation of Young's modulus of single-walled carbon nanotubes by molecular dynamics. *Physics B-Condensed Matter*, 352(1-4):156–163, Oct 2004.

[16] M. Allen. Introduction to molecular dynamics simulation. Technical report, John von Neumann Institute for Computing, 2004.

[17] R. Ansari, B. Motevalli, A. Montazeri, and S. Ajori. Fracture analysis of mono-layer graphene sheets with double vacancy defects via MD simulation. *Solid State Communications*, 151(17):1141–1146, Sep 2011.

[18] V. Nam Do and P. Dollfus. Effects of charged impurities and lattice defects on transport properties of nanoscale graphene structures. *Journal of Applied Physics*, 106(2), Jul 2009.

[19] A. Gulans, A.V. Krasheninnikov, M.J. Puska, and R.M. Nieminen. Bound and free self-interstitial defects in graphite and bilayer graphene: A computational study. *Physical Review B*, 84(2), Jul 2011.

[20] G.D. Lee, C.Z. Wang, E. Yoon, N.M. Hwang, D.Y. Kim, and K.M. Ho. Diffusion, coalescence, and reconstruction of vacancy defects in graphene layers. *Physical Review Lettters*, 95(20), Nov 2005.

[21] G.D. Lee, C.Z. Wang, E. Yoon, N.M. Hwang, and K.M. Ho. Vacancy defects and the formation of local haeckelite structures in graphene from tight-binding molecular dynamics. *Physical Review B*, 74(24), Dec 2006.

[22] M.A.H. Vozmediano, M.P. Lopez-Sancho, T. Stauber, and F Guinea. Local defects and ferromagnetism in graphene layers. *Physical Review B*, 72(15), Oct 2005.

[23] M.P. Ariza and M. Ortiz. Discrete dislocations in graphene. *Journal of the Mechanics and Physics of Solids*, 58(5):710–734, May 2010.

[24] F. Juan, A. Cortijo, and M.A.H. Vozmediano. Dislocations and torsion in graphene and related systems. *Nuclear Physics B*, 828(3):625–637, Apr 2010.

[25] O.V. Yazyev and S.G. Louie. Topological defects in graphene: dislocations and grain boundaries. *Physical Review B*, 81(19), May 2010.

[26] M. Orlita, C. Faugeras, J. Borysiuk, J.M. Baranowski, W. Strupinski, M. Sprinkle, C. Berger, W.A. de Heer, D.M. Basko, G. Martinez, and M. Potemski. Magneto-optics of bilayer inclusions in multilayered epitaxial graphene on the carbon face of SiC. *Physical Review B*, 83(12), Mar 2011.

[27] H.M. Shodja, L. Pahlevani, and E. Hamed. Inclusion problems associated with thin fcc films: Linkage between eigenstrain and inter-atomic potential. *Mechanics of Materials*, 39(8):803–818, Aug 2007.

[28] J. Tersoff. Empirical interatomic potential for carbon, with applications to amorphous-carbon. *Physical Review Letters*, 61(25):2879–2882, Dec 1988.

[29] G.C. Abell. Empirical Chemical Pseudopotential Theory of Molecular and Metallic Bonding. *Physical Review B*, 31(10):6184–6196, May 1985.

[30] J. Tersoff. Empirical interatomic potential for silicon with improved elastic properties. *Physical Review B*, 38(14):9902–9905, Nov 1988.

[31] D.W. Brenner. Empirical potential for hydrocarbons for use in simulating the chemical vapor-deposition of diamond films. *Physical Review B*, 42(15):9458–9471, Nov 1990.

[32] D.W. Brenner, O.A. Shenderova, J.A. Harrison, S.J. Stuart, B. Ni, and SB Sinnott. A second-generation reactive empirical bond order (REBO) potential energy expression for hydrocarbons. *Journal of Physcis-Condensed Matter*, 14(4):783–802, Feb 2002.

[33] L. Lindsay and D.A. Broido. Optimized Tersoff and Brenner empirical potential parameters for lattice dynamics and phonon thermal transport in carbon nanotubes and graphene. *Physical Review B*, 81, May 2010.

APPENDICES

APPENDIX A

ONE-DIMENSIONAL MATLAB CODE

OneD_MDSystem.m- Initializes parameters and sets up the one dimensional chain used in simulation. It also recieves user input to determine which boundary condition is used, and executes code accordingly.

```
function OneD_MDSystem

Iflag=0;
while( ~Iflag )
    disp('0: No Boundary Condition');
    disp('1: Fixed end atoms');
    disp('2: Walls');
    disp('3: Periodic');
    bc = input('Enter Boundary Condition:\n');

    if(bc==0||bc==1||bc==2||bc==3)
        Iflag=1;
    else
        disp('Invalid Option. Try again.');
    end
end

Iflag = 0;
while( ~Iflag )
    disp('0: Without neighbor list construction');
    disp('1: With neighbor list construction');
    NLconst = input('Use neighbor list construction?\n');

    if(NLconst==0||NLconst==1)
        Iflag=1;
    else
        disp('Invalid Option. Try again.');
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%
% Parameter Definitions %
%%%%%%%%%%%%%%%%%%%%%%%%%

n = 50;
m = 1;
```

```
sigma = 1;
epsilon = 1;
beta = 1;
t0 = 0;
tf = 100;
req = 2^(1/6);
areq = .8;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Neighbor list definitions %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if( NLconst )
    Rc = 5;
    NLrc = 20;
    NLc = tf/NLrc;
end

if(bc==0) % No boundary condition without neighbor list construction

    N=501;
    T=linspace(t0,tf,N);

    X = linspace(0,(n-1)*areq*req,n);
    V = zeros(1,n);
    Z = [X V];

    options=odeset('RelTol',1.e-6,'AbsTol',1.e-6);

    if(NLconst)
        [tt,Y]=ode45(@(t,X) LJODE_noBC_NL(t,X,sigma,beta,epsilon,...
                                m,n,Rc,NLc,tf),T,Z,options);
    else
        [tt,Y]=ode45(@(t,X) LJODE_noBC(t,X,sigma,beta,epsilon,...
                                m,n),T,Z,options);
    end

    hold;
    for i = 1:n
        figure(1)
        plot(T, Y(:,i), 'b-');
        xlabel('Time, t (seconds)');
        ylabel('Position');
    end
end

if(bc==1) % fixed end atoms boundary condition

    N=501;
    T=linspace(t0,tf,N);

    X = linspace(0,(n-1)*areq*req,n);
    V = zeros(1,n);

    % fixed boundary condition definitions
    V(1,1)=0; V(1,n)=0;
    Z = [X V];

    options=odeset('RelTol',1.e-6,'AbsTol',1.e-6);

    if(NLconst)
        [tt,Y]=ode45(@(t,X) LJODE_fixed_NL(t,X,sigma,beta,epsilon,...
                              m,n,Rc,NLc,tf),T,Z,options);
    else
        [tt,Y]=ode45(@(t,X) LJODE_fixed(t,X,sigma,beta,epsilon,...
                              m,n),T,Z,options);
    end

    hold;
    for i = 1:n
        figure(1)
        plot(T, Y(:,i), 'b-');
        xlabel('Time, t (seconds)');
        ylabel('Position');
    end
end

if(bc==2) % walls boundry condition

    dN = 10;
    N=tf*dN+1;
    T=linspace(t0,tf,N);

    % wall definitions
    k = 0; % wall hardness; ~1 wall is hard, ~0 wall is soft
    WB = -10;
    WT = 30;
    WBM = WB + 1e-12;
    WTM = WT - 1e-12;

    X = linspace(0,(n-1)*areq*req,n);
    V = zeros(1,n);
    Z = [X V];

    options=odeset('RelTol',1.e-8,'AbsTol',1.e-8,...
 'events', @wallevents);

    if(NLconst)
        [ttf,YF,TE,YE,IE]=ode45(@(t,X) LJODE_walls_NL(t,X,..
                    sigma,beta,epsilon,m,n,WBM,WTM,Rc,tf,...
                    NLc),T,Z,options);
    else
        [ttf,YF,TE,YE,IE]=ode45(@(t,X) LJODE_walls(t,X,sigma,beta,...
                               epsilon,m,n,WBM,WTM),T,Z,options);
    end

    while(numel(ttf)~=N && TE<tf)

        if  any(IE==1)
            YE(1)=WBM;
        end
        if any(IE==n)
            YE(n)=WTM;
        end

        [ts, Nnew] = timestep(TE,ttf,N,dN);
        ttf = ttf(1:end-1);
        YF = YF(1:end-1,:);
        YE(IE+n) = -k*YE(IE+n);
        T=[TE linspace(ts,tf,Nnew)];
        Z=YE;

        if(numel(T) == 1)
            T = [T T+1/dN];
        end

        if(NLconst)
            [tt,Y,TE,YE,IE]=ode45(@(t,X) LJODE_walls_NL(t,X,...
                        sigma,beta,epsilon,m,n,WBM,WTM,...
                        Rc,tf,NLc),T,Z,options);
        else
            [tt,Y,TE,YE,IE]=ode45(@(t,X) LJODE_walls(t,X,...
                        sigma,beta,epsilon,m,n,WBM,WTM),T,Z,options);
        end

        ttf = [ttf; tt(2:end)];
        YF = [YF; Y(2:end,:)];

    end

    hold;
    for i = 1:n
        figure(1)
        plot(ttf, YF(:,i), 'b-', ttf, WB, 'k-', ttf, WT, 'k-');
        xlabel('Time, t (seconds)');
        ylabel('Position, x');
        axis([0 tf WB-5 WT+5])
    end

end

% Wall events nested function for the wall boundary condition
function [value, isterminal, direction] = wallevents(t,y)
        value = (y(1:n)-WT).*(y(1:n)-WB);
        isterminal(1:n) = 1;
        direction=0;
```

```
end

if(bc==3) % periodic boundary condition

    N=101;
    T=linspace(t0,tf,N);

    rc = 4;
    aeq=2^(1/6);

    X = linspace(aeq,n*aeq,n);
    V = zeros(1,n);
    Z = [X V];

    L = X(n);

    options=odeset('RelTol',1.e-6,'AbsTol',1.e-6);

    if(NLconst)
        [tt,Y]=ode45(@(t,X) LJODE_periodic_NL(t,X,...
                        sigma,beta,epsilon,m,n,rc,L),T,Z,options);
    else
        k = 10; % periodicity range
        PL = periodicList(n,k);
        [tt,Y]=ode45(@(t,X) LJODE_periodic(t,X,...
                        sigma,beta,epsilon,m,n,k,PL,L),T,Z,options);
    end

    hold;
    for i = 1:n
        figure(1)
        plot(T, Y(:,i), 'b-');
        xlabel('Time, t (seconds)');
        ylabel('Position');
    end
end

end
```

LJODE_noBC.m- ODE function used in the case of no boundary conditions without neighbor list construction.

```
function [X] = LJODE_noBC(t,y,sigma,beta,epsilon,m,n)

X = zeros(2*n,1);

for i=1:n
    X(i) = y(i+n);
    for j=1:n
        if(j~=i)
```

```
                X(i+n) = X(i+n)+ ForceLJ(y(i)-y(j),sigma,epsilon)/m;
            end
        end

        X(i+n) = X(i+n) - beta/m*y(i+n);

    end
```

LJODE_noBC_NL.m- ODE function used in the case of no boundary conditions with neighbor list construction.

```
function [X] = LJODE_noBC_NL(t,y,sigma,beta,epsilon,m,n,Rc,NLc,tf)

persistent NL NLval

if t==0
    NL = neighborList(y,n,Rc);
    NLval = 0;
end

if(mod(floor(t),NLc)==0 && floor(t)~=tf)

    if(floor(t) > NLval)
       disp(['Reconstructing Neighbor List at t=',num2str(floor(t))]);
       NL = neighborList(y,n,Rc);
       NLval = floor(t);
    end

end

[NLrows NLcols] = size(NL);
X = zeros(2*n,1);

for k = 1:NLrows

    i = NL(k,1);
    j = NL(k,2);

    X(i) = y(i+n);
    X(j) = y(j+n);

    X(i+n) = X(i+n)+ ForceLJ(y(i)-y(j),sigma,epsilon)/m;
    X(j+n) = X(j+n)- ForceLJ(y(i)-y(j),sigma,epsilon)/m;

end

for i = 1:n
    X(i+n) = X(i+n) - beta/m*y(i+n);
end
```

LJODE_fixed.m- ODE function used in the case of fixed ends boundary condition without neighbor list construction.

```
function [X] = LJODE_fixed(t,y,sigma,beta,epsilon,m,n)

X = zeros(2*n,1);

for i=2:n-1
    X(i) = y(i+n); % dr/dt = v
    for j=1:n
        if(j~=i)
            X(i+n) = X(i+n)+ ForceLJ(y(i)-y(j),sigma,epsilon)/m;
        end
    end

    X(i+n) = X(i+n) - beta/m*y(i+n);

end
```

LJODE_fixed_NL.m- ODE function used in the case of fixed ends boundary condition with neighbor list construction.

```
function [X] = LJODE_fixed_NL(t,y,sigma,beta,epsilon,m,n,Rc,NLc,tf)

persistent NL NLval

if t==0
    NL = neighborList(y,n,Rc); % construct NL at first time step
    NLval = 0;
end

if(mod(floor(t),NLc)==0 && floor(t)~=tf)

    if(floor(t) > NLval)
       disp(['Reconstructing Neighbor List at t=',num2str(floor(t))]);
       NL = neighborList(y,n,Rc);
       NLval = floor(t);
    end

end

[NLrows NLcols] = size(NL);
X = zeros(2*n,1);

for k = 1:NLrows

    i = NL(k,1);
    j = NL(k,2);
    if((i~=1 && i~=n) && (j~=1 && j~=n))

        X(i) = y(i+n);
        X(i+n) = X(i+n)+ ForceLJ(y(i)-y(j),sigma,epsilon)/m;

        X(j) = y(j+n); % dr/dt = v
        X(j+n) = X(j+n)- ForceLJ(y(i)-y(j),sigma,epsilon)/m;

    elseif((i==1 || i==n) && (j~=1 && j~=n))

        X(j) = y(j+n);
        X(j+n) = X(j+n)- ForceLJ(y(i)-y(j),sigma,epsilon)/m;

    elseif((i~=1 || i~=n) && (j==1 || j==n))

        X(i) = y(i+n);
        X(i+n) = X(i+n)+ ForceLJ(y(i)-y(j),sigma,epsilon)/m;

    end

end

for i=1:n
    X(i+n) = X(i+n) - beta/m*y(i+n);
end
```

LJODE_walls.m- ODE function used in the case of walls boundary condition without neighbor list construction.

```
function [X] = LJODE_walls(t,y,sigma,beta,epsilon,m,n,WBM,WTM)

X = zeros(2*n,1); % initialize ODE solution vector

for i=1:n

    if(i==1)
        X(i) = y(i+n);
        Force = 0;
        for j=1:n
            if(j~=i)
                Force = Force + ForceLJ(y(i)-y(j),sigma,epsilon)/m;
            end
        end

        if(~(y(i) ==WBM && Force<0 && y(i+n) ==0))
            X(i+n) = X(i+n)+Force - beta/m*y(i+n);
        end
    elseif(i==n)
        X(i) = y(i+n);
        Force = 0;
        for j=1:n
            if(j~=i)
                Force = Force + ForceLJ(y(i)-y(j),sigma,epsilon)/m;
            end
        end

        if(~(y(i) ==WTM && Force>0 && y(i+n) ==0))
            X(i+n) = X(i+n)+Force - beta/m*y(i+n);
        end

    else

        X(i) = y(i+n); % dr/dt = v
        for j=1:n
            if(j~=i)
                X(i+n) = X(i+n)+ ForceLJ(y(i)-y(j),sigma,epsilon)/m;
            end
        end

        X(i+n) = X(i+n) - beta/m*y(i+n);
    end
end
```

LJODE_walls_NL.m- ODE function used in the case of walls boundary condition with neighbor list construction.

```
function [X] = LJODE_walls_NL(t,y,sigma,beta,epsilon,m,n,...
                                  WBM,WTM,Rc,tf,NLc)

persistent NL NLval

if t==0
    NL = neighborList(y,n,Rc);
    NLval = 0;
end

if(mod(floor(t),NLc)==0 && floor(t)~=tf)

    if(floor(t) > NLval)
       disp(['Reconstructing Neighbor List at t=',num2str(floor(t))]);
       NL = neighborList(y,n,Rc);
       NLval = floor(t);
    end

end
```

```
X = zeros(2*n,1);
[NLrows NLcols] = size(NL);

for c=1:NLrows

    i = NL(c,1);
    j = NL(c,2);

    if(i==1)
        X(i) = y(i+n);
        X(j) = y(j+n);
        Force = 0;
        Force = Force + ForceLJ(y(i)-y(j),sigma,epsilon)/m;

        if(~(y(i) ==WBM && Force<0 && y(i+n) ==0))
            X(i+n) = X(i+n)+Force;
            X(j+n) = X(j+n)-Force;
        end

    elseif(i==n)
        X(i) = y(i+n);
        X(j) = y(j+n);
        Force = 0;
        Force = Force + ForceLJ(y(i)-y(j),sigma,epsilon)/m;

        if(~(y(i) ==WTM && Force>0 && y(i+n) ==0))
            X(i+n) = X(i+n)+Force;
            X(j+n) = X(j+n)-Force;
        end

    else
        X(i) = y(i+n);
        X(j) = y(j+n);
        X(i+n) = X(i+n)+ ForceLJ(y(i)-y(j),sigma,epsilon)/m;
        X(j+n) = X(j+n)- ForceLJ(y(i)-y(j),sigma,epsilon)/m;
    end
end

for i = 1:n
    X(i+n) = X(i+n) - beta/m*y(i+n);
end
```

---

timestep.m- Function used to calculate the next time step in the walls boundary condition simulations.

```
function [ts,Nnew] = timestep(TE,ttf,N,dN)

f = floor(TE);
c = ceil(dN*(TE-f));
```

```
ts = c/dN+f;

Nnew = N-numel(ttf)+1;
```

LJODE_periodic.m- ODE function used in the case of periodic boundary condition without neighbor list construction.

```
function [X] = LJODE_periodic(t,y,sigma,beta,epsilon,m,n,k,PL,L)
t
X = zeros(2*n,1);
[PLrows PLcols] = size(NL);

for c=1:PLrows

    i = PL(c,1);  % i is atom 1
    j = PL(c,2);  % j is atom 2

    if(j==0)
        j=n;
    end

    X(i) = y(i+n);
    X(j) = y(j+n);
    d = PeriodicDistance(i,j,n,k,y,L);

    X(i+n) = X(i+n)- ForceLJ(d,sigma,epsilon)/m;
    X(j+n) = X(j+n)+ ForceLJ(d,sigma,epsilon)/m;

end

for i = 1:n
    X(i+n) = X(i+n) - beta/m*y(i+n);
end
```

---

periodicList.m- Function used to create periodic list in the periodic condition without neighbor list construction case.

```
function [NeighborList] = periodicList(n,k)

NeighborList = [];

for i=1:n
    for j = 1:k

        x = [i,mod(i+j,n)];
        y = [i,mod(i-j,n)];
        isXduplicate = 0;
        isYduplicate = 0;

        [NeighborListRows, NeighborListColumns] = size(NeighborList);
        for c = 1:NeighborListRows

            duplicate = [NeighborList(c,2),NeighborList(c,1)];

            if(x == duplicate)
                isXduplicate = 1;
            end
            if( y ==duplicate)
                isYduplicate = 1;
            end
        end

        if(~isXduplicate && ~isYduplicate)
            NeighborList(NeighborListRows+1,:) = x;
            NeighborList(NeighborListRows+2,:) = y;
        elseif(~isXduplicate && isYduplicate)
            NeighborList(NeighborListRows+1,:) = x;
        elseif(isXduplicate && ~isYduplicate)
            NeighborList(NeighborListRows+1,:) = y;
        end

    end
end

% Sort Neighbor List
s = 1;
e = 1;
c = 1;
[NeighborListRows NeighborListCol] = size(NeighborList);

for i = 1:n

    while(c<=NeighborListRows && NeighborList(c,1)==i)
        e = e+1;
        c = c+1;
    end

    NeighborList(s:e-1,2)=sort(NeighborList(s:e-1,2));
    s = e;

end
```

PeriodicDistance.m- Function used to calculate wrap around distances in the periodic condition without neighbor list construction case.

```
function d = PeriodicDistance(i,j,n,k,y,L)

wrapback =0;
wrapforward=0;

if(i-j>k)
    wrapforward = 1;
elseif(i-j<-k)
    wrapback = 1;
end

% calculate distance
if (wrapback)
    d = y(j)-(y(i)+L);
elseif(wrapforward)
    d = -y(j) -(L-y(i));
else
    d = y(j) - y(i);
end
```

LJODE_periodic_NL.m- ODE function used in the case of periodic boundary condition with neighbor list construction.

```
function [X] = LJODE_periodic_NL(t,y,sigma,beta,epsilon,m,n,rc,L)

NL = PeriodicNeighborList(n,rc,y,L);
X = zeros(2*n,1);
[NLrows NLcols] = size(NL);

for c=1:NLrows

    if(mod(c,2)==1) % c is odd, then force is backward
        d = NL(c,3);
        i = NL(c,1);
        X(i) = y(i+n);
        X(i+n) = X(i+n)- ForceLJ(d,sigma,epsilon)/m;
    elseif(mod(c,2)==0) % if c is even, then force is forward
        d = NL(c,3);
        i = NL(c,1);
        X(i) = y(i+n);
        X(i+n) = X(i+n)+ ForceLJ(d,sigma,epsilon)/m;
    end

end

for i = 1:n
    X(i+n) = X(i+n) - beta/m*y(i+n);
end
```

PeriodicNeighborList- Function to construct the neighbor list in the periodc boundary condition case with neighbor list construction.

```
function [NeighborList] = PeriodicNeighborList(n,rc,y,L)

NeighborList = [];

for i = 1:n

    distance=0;
    jforward=i+1;
    jback = i-1;

    while(distance<rc)
        % find forward atoms for neighbor list
        if jforward>n
            distance = L + y(jforward-n) - y(i);
        else
            distance = y(jforward) - y(i);
        end

        NeighborList(end+1,:) = [i,jforward,distance];
        jforward = jforward+1;

        % find backward atoms for neighbor list
        if jback<1
            distance = y(i) + L - y(jback+n);
        else
            distance = y(i) - y(jback);
        end

        NeighborList(end+1,:)= [i,jback,distance];
        jback = jback -1;

    end
end
```

---

neighborList.m- Function used to construct the neighbor list in the no boundary condition, fixed ends condition, and walls condition with neighbor list construction.

```
function [NL] = neighborList(y,n,Rc)

c=1;
for i = 1:n
    for j=i+1:n

        if(y(j)-y(i)<= Rc)
            NL(c,:) = [i,j];
            c = c+1;
        end

    end
end
```

ForceLJ.m- Function used to calculate the Lennard Jones force for a given pariwise interaction that is used in all MD cases.

```
function [f] = ForceLJ(d, sigma, epsilon)

f = epsilon*(48*sigma^12/d^13 -24*sigma^6/d^7);
```

APPENDIX B

TWO-DIMENSIONAL MATLAB CODE

TwoD_MDSystem.m- Initializes parameters and sets up the two dimensional atomic lattice. It also recieves user input to define which boundary condition is used in the simulation, as well as the defect to be implemented, then executes code accordingly.

```
function TwoD_MDSystem
tic;

%%%%%%%%%%%%%%%%%%%%%%%
% LATTICE DEFINITIONS %
%%%%%%%%%%%%%%%%%%%%%%%

r = 24; % atomic number of rows
c = 40; % number of atomic columns
epsreq = 1; % uniform perturbation from equilibrium spacing
perturb = 0; % random non uniform perturbation
defect = 'vacancy';
xbc = 'Periodic';
ybc = 'NoBC';

onBoundary = 0;
extendBoundary = 0;

%%%%%%%%%%%%%%%%%%%%%%
% ATOMIC DEFINITIONS %
%%%%%%%%%%%%%%%%%%%%%%

m = 1;
epsilon = 1;
sigma = 1;
beta = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ATOMIC SPACING DEFINITIONS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cflag = 'case5k';
req = 2^(1/6);
areq = 1;

%%%%%%%%%%%%%%%%%%%%%%%%
% TIMESTEP DEFINITIONS %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%

t0 = 0;
tf = 1000;
dN = 100;
N=tf*dN+1;
T = linspace(t0,tf,N);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% NEIGHBOR LIST DEFINITIONS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Rc = 5;
NLrc = 100;  % number of neigborlist reconstructions
NLc = tf/NLrc;

%%%%%%%%%%%%%%%%%%%%%%%%%%
% LATTICE CONSTRUCTION %
%%%%%%%%%%%%%%%%%%%%%%%%%%

[xx xy n] = ConstructLattice(r,c,epsreq, perturb);
if(strcmp(defect,'vacancy'))
    [xx xy n] = CreateVacancy(xx,xy,n);
elseif(strcmp(defect,'inclusion'))
    [xx xy n] = CreateInterstitial(xx,xy,n);
elseif(strcmp(defect,'dislocation'))
    [xx xy n] = CreateDislocation(xx,xy,n,r,c);
end

%%%%%%%%%%%%%%%%%%%%%%%%
% CENTER AROUND ORIGIN %
%%%%%%%%%%%%%%%%%%%%%%%%

Lx = max(xx);
xx = xx - Lx/2;
Ly = max(xy);
xy = xy - Ly/2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ATOMIC VELOCITY DEFINITIONS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

vx = zeros(1,n);
vy = zeros(1,n);
Z = [xx vx xy vy];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BOUNDARY CONDITION DEFINITIONS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% FIXED BOUNDARY CONDITION DEFINITIONS
Fixed_Atoms = 0;
```

```
if(strcmp(xbc,'Fixed')) % fixed in x

   min_xx = min(xx);
   max_xx = max(xx);

   for i = 1:numel(xx)
       if(xx(i) == min_xx || xx(i) == max_xx)
           Fixed_Atoms = [Fixed_Atoms i];
       end
   end

end
if(strcmp(ybc,'Fixed')) %fixed in y

    min_xy = min(xy);
    max_xy = max(xy);

  for i = 1:numel(xy)
       if(xy(i) == min_xy || xy(i)== max_xy)
           Fixed_Atoms = [Fixed_Atoms i];
       end
   end

end

Fixed_Atoms(1) = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% WALLS BOUNDARY CONDITION DEFINITIONS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

k = 1;
WA = 1e-12;
if(strcmp(xbc,'Walls')||strcmp(ybc,'Walls'))
    WL = -Lx/2 - (.5*req)*.5 ;
    WR = Lx/2 + (.5*req)*.5;
    WLM = WL + WA;
    WRM = WR - WA;
    WB = -Ly/2 - (sqrt(3)/2)*req*.5;
    WT = Ly/2 + (sqrt(3)/2)*req*.5;
    WBM = WB + WA;
    WTM = WT - WA;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PERIODIC BOUNDARY CONDITIONS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LA = 1e-12;
if(strcmp(xbc,'Periodic'))
    if(extendBoundary)
        Lx = Lx + 2*(.5*req);
    else
        Lx = Lx + .5*req;
    end
end
if(strcmp(ybc,'Periodic'))
    if(extendBoundary)
        Ly = Ly + 2*(sqrt(3)/2)*req;
    else
        Ly = Ly + (sqrt(3)/2)*req;

    end
end

if(onBoundary)
    xx = xx - .5*(.5*req);
    xy = xy - .5*(sqrt(3)/2)*req;
    Z = [xx vx xy vy];
end

hold;
for i=1:n
    plot(xx(i),xy(i),'bo','MarkerEdgeColor','b',...
                       'MarkerFaceColor','b',...
                       'MarkerSize',2)
    text(xx(i),xy(i),num2str(i));
    title('Initial Positions');
    xlabel('X');
    ylabel('Y');
    axis equal
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LJODE FUNCTION DEFINITIONS, MD SIMULATION %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pause;

if(strcmp(xbc,'NoBC')&& strcmp(ybc,'NoBC'))

    options = odeset('RelTol',1.e-6,'AbsTol',1.e-6);
    [ttf,YF]=ode45(@(t,X)LJODE_NoBC(t,X,n,m,epsilon,sigma,...
                         beta,Rc,NLc,tf,cflag),T,Z,options);

elseif((strcmp(xbc,'NoBC')&& strcmp(ybc,'Fixed'))||...
       (strcmp(xbc,'Fixed')&& strcmp(ybc,'NoBC'))||...
       (strcmp(xbc,'Fixed')&& strcmp(ybc,'Fixed')))

    options = odeset('RelTol',1.e-6,'AbsTol',1.e-6);
    [ttf,YF]=ode45(@(t,X)LJODE_Fixed(t,X,n,m,epsilon,sigma,...
                         beta,Fixed_Atoms,Rc,NLc,tf),T,Z,options);

elseif((strcmp(xbc,'NoBC')&&strcmp(ybc,'Walls'))||...
       (strcmp(xbc,'Walls')&&strcmp(ybc,'NoBC'))||...
       (strcmp(xbc,'Walls')&&strcmp(ybc,'Walls')))

    options = odeset('RelTol',1.e-6,'AbsTol',1.e-6,...
                     'events',@wallEvents);
    [ttf,YF,TE,YE,IE]=ode45(@(t,X)LJODE_Walls(t,X,n,m,epsilon,...
                           sigma,beta,Rc,NLc,tf,xbc,ybc,...
                           WTM,WBM,WLM,WRM),T,Z,options);

elseif((strcmp(xbc,'NoBC')&&strcmp(ybc,'Periodic'))||...
       (strcmp(xbc,'Periodic')&&strcmp(ybc,'NoBC'))||...
       (strcmp(xbc,'Periodic')&&strcmp(ybc,'Periodic')))

options = odeset('RelTol',1.e-6,'AbsTol',1.e-6,...
                         'events',@periodicEvents);
     [ttf,YF,TE,YE,IE]=ode45(@(t,X)LJODE_Periodic(t,X,n,m,...
                           epsilon,sigma,beta,Lx,Ly,...
                           xbc,ybc,Rc,NLc,tf),T,Z,options);

elseif((strcmp(xbc,'Fixed')&&strcmp(ybc,'Walls'))||...
       (strcmp(xbc,'Walls')&&strcmp(ybc,'Fixed')))

        options = odeset('RelTol',1.e-6,'AbsTol',1.e-6,...
                         'events',@wallEvents);
        [ttf,YF,TE,YE,IE]=ode45(@(t,X)LJODE_FixedWalls(t,X,n,m,...
                           epsilon,sigma,beta,Fixed_Atoms,...
                           Rc,NLc,tf,xbc,ybc,...
                           WTM,WBM,WLM,WRM),T,Z,options);

elseif((strcmp(xbc,'Fixed')&&strcmp(ybc,'Periodic'))||...
       (strcmp(xbc,'Periodic')&&strcmp(ybc,'Fixed')))

        options = odeset('RelTol',1.e-8,'AbsTol',1.e-8,...
                         'events',@periodicEvents);
        [ttf,YF,TE,YE,IE]=ode45(@(t,X)LJODE_FixedPeriodic(t,X,n,m,...
                           epsilon,sigma,beta,Fixed_Atoms,...
                           Lx,Ly,xbc,ybc,Rc,NLc,tf),T,Z,options);

elseif((strcmp(xbc,'Walls')&&strcmp(ybc,'Periodic'))||...
       (strcmp(xbc,'Periodic')&&strcmp(ybc,'Walls')))

        options = odeset('RelTol',1.e-6,'AbsTol',1.e-6,...
                         'events',@wall_periodic_Events);
        [ttf,YF,TE,YE,IE]=ode45(@(t,X)LJODE_WallsPeriodic(t,X,n,m,...
                           epsilon,sigma,beta,...
                           Lx,Ly,xbc,ybc,Rc,NLc,...
                           tf,WTM,WBM,WLM,WRM),T,Z,options);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BREAK FOR WALLS OR PERIODIC EVENTS FUNCTIONS %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while(numel(ttf)~=N && TE(end)<tf)

    IE = IE(end);
    xxe = YE(end,IE); % x position of atom
    xye = YE(end,IE+2*n); % y position of atom
    vxe = YE(end,IE+n); % x velocity of atom
    vye = YE(end,IE+3*n); % y velocity of atom

    if(strcmp(xbc,'Periodic')) %periodic in x
        if(xxe >= Lx/2) %right periodic box
            YE(end,IE) = YE(end,IE)-Lx+LA;
        end
        if(xxe <= -Lx/2) %left periodic box
            YE(end,IE) = YE(end,IE)+Lx-LA;
        end
    end
    if(strcmp(ybc,'Periodic')) %periodic in y
        if(xye >= Ly/2) % top periodic box
            YE(end,IE+2*n) = YE(end,IE+2*n)-Ly+LA;
        end
        if(xye <= -Ly/2)% bottom periodic box
            YE(end,IE+2*n) = YE(end,IE+2*n)+Ly-LA;
        end
    end
    if(strcmp(xbc,'Walls')) % walls in x
        if( xxe>=WL-WA && xxe<=WL+WA )
            YE(end,IE) = WLM;
            YE(end,IE+n) = -k*vxe;
            YE(end,IE+3*n) = vye;
        end
        if( xxe>=WR-WA && xxe<=WR+WA )
            YE(end,IE) = WRM;
            YE(end,IE+n) = -k*vxe;
            YE(end,IE+3*n) = vye;
        end
    end
    if(strcmp(ybc,'Walls')) % walls in y
        if( xye>=WT-WA && xye<=WT+WA ) %top wall
            YE(end,IE+2*n) = WTM;
            YE(end,IE+n) = vxe;
            YE(end,IE+3*n) = -k*vye;
        end
        if( xye>=WB-WA && xye<=WB+WA ) %bottom wall
            YE(end,IE+2*n) = WBM;
            YE(end,IE+n) = vxe;
            YE(end,IE+3*n) = -k*vye;
        end
    end

   [ts, Nnew] = timestep(TE(end),ttf,N,dN);
```

```
    ttf = ttf(1:end-1);
    YF = YF(1:end-1,:);

    T=[TE(end) linspace(ts,tf,Nnew)];
    Z=YE(end,:);

    clear TE IE YE;

    if((strcmp(xbc,'NoBC')&&strcmp(ybc,'Walls'))||...
       (strcmp(xbc,'Walls')&&strcmp(ybc,'NoBC'))||...
       (strcmp(xbc,'Walls')&&strcmp(ybc,'Walls')))

        [tt,Y,TE,YE,IE]=ode45(@(t,X)LJODE_Walls(t,X,n,m,...
                        epsilon,sigma,beta,Rc,NLc,tf,...
                        xbc,ybc,WTM,WBM,WLM,WRM),T,Z,options);

    elseif((strcmp(xbc,'NoBC')&&strcmp(ybc,'Periodic'))||...
           (strcmp(xbc,'Periodic')&&strcmp(ybc,'NoBC'))||...
           (strcmp(xbc,'Periodic')&&strcmp(ybc,'Periodic')))

        [tt,Y,TE,YE,IE]=ode45(@(t,X)LJODE_Periodic(t,X,n,m,...
                        epsilon,sigma,beta,Lx,Ly,...
                        xbc,ybc,Rc,NLc,tf),T,Z,options);

    elseif((strcmp(xbc,'Fixed')&&strcmp(ybc,'Walls'))||...
           (strcmp(xbc,'Walls')&&strcmp(ybc,'Fixed')))

        [tt,Y,TE,YE,IE]=ode45(@(t,X)LJODE_FixedWalls(t,X,n,m,...
                        epsilon,sigma,beta,Fixed_Atoms,...
                        Rc,NLc,tf,xbc,ybc,WTM,...
                        WBM,WLM,WRM),T,Z,options);

    elseif((strcmp(xbc,'Fixed')&&strcmp(ybc,'Periodic'))||...
           (strcmp(xbc,'Periodic')&&strcmp(ybc,'Fixed')))

        [tt,Y,TE,YE,IE]=ode45(@(t,X)LJODE_FixedPeriodic(t,X,n,m,...
                        epsilon,sigma,beta,Fixed_Atoms,...
                        Lx,Ly,xbc,ybc,Rc,NLc,tf),T,Z,options);

    elseif((strcmp(xbc,'Walls')&&strcmp(ybc,'Periodic'))||...
           (strcmp(xbc,'Periodic')&&strcmp(ybc,'Walls')))

        [tt,Y,TE,YE,IE]=ode45(@(t,X)LJODE_WallsPeriodic(t,X,n,m,...
                        epsilon,sigma,beta,Lx,Ly,xbc,ybc,...
                        Rc,NLc,tf,WTM,WBM,WLM,WRM),T,Z,options);

    end

    ttf = [ttf; tt(2:end)];
    YF = [YF; Y(2:end,:)];

end
```

```
%%%%%%%%%%%%%%%%%%%%%%
% SAVING DEFINITIONS %
%%%%%%%%%%%%%%%%%%%%%%

savefile = ['XBoundary_Condition_' cflag];
save(savefile, 'xbc');

savefile = ['YBoundary_Condition_' cflag];
save(savefile, 'ybc');

savefile = ['Initial_Positions_' cflag ];
initialpositions= [xx xy];
save(savefile, 'initialpositions')

savefile = ['Final_Positions_' cflag ];
finalpositions= [YF(end,1:n) YF(end, 2*n+1:3*n)];
save(savefile, 'finalpositions');

savefile = ['Data_Positions_' cflag ];
datapositions = [YF(:,1:n) YF(:,2*n+1:3*n)];
save(savefile, 'datapositions');

if(strcmp(ybc,'Walls'))
    savefile = ['Walls_Y_' cflag];
    wallsY = [WB WT];
    save(savefile, 'wallsY')
end
if(strcmp(xbc,'Walls'))
    savefile = ['Walls_X_' cflag];
    wallsX = [WL WR];
    save(savefile, 'wallsX')
end
if(strcmp(ybc,'Periodic'))
    savefile = ['Periodic_Y_' cflag];
    save(savefile, 'Ly');
end
if(strcmp(xbc,'Periodic'))
    savefile = ['Periodic_X_' cflag];
    save(savefile, 'Lx');
end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % EVENTS FUNCTIONS DEFINITIONS %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   function [value, isterminal, direction] = wallEvents(t,X)
        value = (X(1:n)-WL).*(X(1:n)-WR).*...
                (X(2*n+1:3*n)-WT).*(X(2*n+1:3*n)-WB);
        isterminal(1:n) = 1;
        direction=0;
   end
   function[value, isterminal, direction] = periodicEvents(t,X)
        value = (X(1:n)-Lx/2).*(X(1:n)+Lx/2).*...
                (X(2*n+1:3*n)-Ly/2).*(X(2*n+1:3*n)+Ly/2);
        isterminal(1:n,1) = 1;
        direction(1:n,1)=0;
   end
   function [value, isterminal, direction] = wall_periodic_Events(t,X)
        value = (X(1:n)-WL).*(X(1:n)-WR).*(X(2*n+1:3*n)-WT).*...
                (X(2*n+1:3*n)-WB).*(X(1:n)-Lx/2).*(X(1:n)+Lx/2).*...
                (X(2*n+1:3*n)-Ly/2).*(X(2*n+1:3*n)+Ly/2);
        isterminal(1:n) = 1;
        direction=0;
   end

end
```

ConstructLattice.m- Function that creates an equilibrated Lennard-Jones lattice, given an r number of rows and c number of columns in the lattice. This funciton also creates either uniform or nonuniform perturbations given *epsreq* and *perturb*, respectively.

```
function [xx,xy,n] = ConstructLattice(r,c,epsreq,perturb)

%ATOMIC STRUCTURE DEFINITIONS
if(mod(c,2)==0) % if even number of columns
    c1 = c/2;
    c2 = c/2;
else
    c1 = (c+1)/2;
    c2 = c1 - 1;
end

if(mod(r,2)==0) % if even number of rows
    n = c1*r/2 + c2*r/2;
else
    n = c1*ceil(r/2) + c2*floor(r/2);
end

% ATOMIC SPACING DEFINITIONS
req = 2^(1/6);
spacing = epsreq*req;
a = .5*spacing;
b = (sqrt(3)/2)*spacing;

% ATOMIC POSITIONS DEFINITIONS
xx = 0;
xy = 0;

for i=1:c
```

```
    clear xpos ypos

    if(mod(i,2)==1)
        atomsincol = ceil((r/2));
        xpos(1:atomsincol) = (i-1)*a;

        for j=1:atomsincol
            ypos(j) = 2*(j-1)*b;
        end

    elseif(mod(i,2)==0)
        atomsincol = floor(r/2);
        xpos(1:atomsincol) = (i-1)*a;

        for j=1:atomsincol
            ypos(j) = (2*j-1)*b;
        end
    end

    xx = [xx xpos];
    xy = [xy ypos];

end

xx(1) = [];
xy(1) = [];

if(perturb)
    xx = xx + .2*(2*rand(1,n) - 1);
    xy = xy + .2*(2*rand(1,n) - 1);
end
```

CreateVacancy.m- Function that creates a vacancy defect in the given lattice, given the x and y positions of atoms. The vacancy is either chosen randomly by automatic construction, or chosen specifically by manual construction.

```
function [xx,xy,n] = CreateVacancy(xx,xy,n)

 hold;
 for i=1:n
     plot(xx(i),xy(i),'bo','MarkerEdgeColor','b',...
                       'MarkerFaceColor','b',...
                       'MarkerSize',2)
     text(xx(i),xy(i),num2str(i));
     title('Initial Positions');
     xlabel('X');
     ylabel('Y');
     axis equal
 end
 manual_or_auto = input('Manual or Auto? 1: Manual 2:Auto\n');

 if(manual_or_auto==2)

     remove_atom = randi(n);
     xx(remove_atom) = [];
     xy(remove_atom) = [];
     n = n-1;

     hold;
     for i=1:n
         plot(xx(i),xy(i),'bo','MarkerEdgeColor','b',...
             'MarkerFaceColor','b',...
             'MarkerSize',2)
         text(xx(i),xy(i),num2str(i));
         title('Initial Positions');
         xlabel('X');
         ylabel('Y');
         axis equal
     end
 end
 if(manual_or_auto==1)

     done = 0;

     while(~done)

         remove_atom = input('Enter atom index to be removed\n');
         xx(remove_atom) = [];
         xy(remove_atom) = [];
         n = n-1;

         close;
         hold;
         for i=1:n
             plot(xx(i),xy(i),'bo','MarkerEdgeColor','b',...
                 'MarkerFaceColor','b',...
                 'MarkerSize',2)
             text(xx(i),xy(i),num2str(i));
             title('Initial Positions');
             xlabel('X');
             ylabel('Y');
             axis equal
         end


         done = input('Done? (0/1)');
     end
 end

end
```

CreateDislocation.m- Funciton the creates a dislocation defect in the given lattice, given the x and y positions of atoms. Any atom in a removal row is selected either randomly or manually, then half of the row containing the atom is removed.

```
function [xx,xy,n] = CreateDislocation(xx,xy,n,r,c)

bottom_atom = 1;
bottom_row = 1;

if(mod(c,2)==0) % if even number of columns
    batoms = c/2;
else
    batoms = (c-1)/2;
end

for i=1:batoms-1

    bottom_atom = bottom_atom + r;
    bottom_row = [bottom_row bottom_atom];

end

remove_atom = 73;
%remove_atom = bottom_row(randi(batoms+1));

baFlag = 1;
atoms_in_removal_row = 0;
count = 1;
while( remove_atom < n && baFlag)

    atoms_in_removal_row(end+1) = remove_atom;

    if(mod(count,2)==0)
        remove_atom = remove_atom + r/2+1;
    else
        remove_atom = remove_atom + r/2;
    end

    for i = 1:numel(bottom_row)
        if(remove_atom==bottom_row(i))
            baFlag = 0;
        end
    end

    count = count+1;
end

atoms_in_removal_row(1)=[];
```

```
numer_atoms_to_remove = floor(numel(atoms_in_removal_row)/2);

RA = atoms_in_removal_row(numer_atoms_to_remove+1:...
                          2*numer_atoms_to_remove);
xx(RA) = NaN;
xy(RA) = NaN;

i = 1;
while( i<=numel(xx) )

    if(isnan(xx(i)) && isnan(xy(i)))
        xx(i) = [];
        xy(i) = [];
        n = n-1;
    end

    i = i+1;
end

n = numel(xx);
```

CreateInterstitial.m- Function that creates an interstitial defect in the given lattice, given the x and y positions of atoms. The interstitial is created by manually defining the left and right adjacent atoms to the interstitial location, then shifting the two atoms and inserting the extra atom.

```
function [xx,xy,n] = CreateInterstitial(xx,xy,n)

hold;
for i=1:n
    plot(xx(i),xy(i),'ko','MarkerEdgeColor','k',...
        'MarkerFaceColor','k',...
        'MarkerSize',2)
    text(xx(i),xy(i),num2str(i));
    title('Initial Positions');
    xlabel('X');
    ylabel('Y');
    axis equal
end

done = 0;

while(~done)

    atom1 = input('Left adjacent atom?');
    atom2 = input('Right adjacent atom?');

    extra_atom_y = xy(atom1);
    space_between_adjacents = abs(xx(atom2)-xx(atom1));
    xx(atom1) = xx(atom1) - space_between_adjacents/4;
    xx(atom2) = xx(atom2) + space_between_adjacents/4;

    extra_atom_x = (abs(xx(atom2)+xx(atom1)))/2;

    %extra_atom_x = randi( floor(max(xx)) );
    %extra_atom_y = randi( floor(max(xy)) );

    xx = [xx extra_atom_x];
    xy = [xy extra_atom_y];

    n = n+1;

    close;
    hold;
    for i=1:n
        plot(xx(i),xy(i),'ko','MarkerEdgeColor','k',...
            'MarkerFaceColor','k',...
            'MarkerSize',2)
        title('Initial Positions');
        xlabel('X');
        ylabel('Y');
        axis equal
    end

    done = input('Done? (0/1)');

end
```

LJODE_noBC.m- ODE function used in the no boundary condition case, where there are no boundary conditions in either the x or y directions.

```
function[Y] = LJODE_NoBC(t,X,n,m,epsilon,sigma,beta,Rc,NLc,tf)

persistent NL NLval

if(t==0)
    NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
    disp('Constructing Neighbor List at t=0');
    NLval = 0;
end

% Neighborlist Reconstruction after defined time interval
if(mod(floor(t),NLc)==0 && floor(t)~=tf)

  if(floor(t) > NLval)
     disp(['Reconstructing Neighbor List at t=', num2str(floor(t))]);
     NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
     NLval = floor(t);
  end

end

[NLrows NLcols] = size(NL);
Y = zeros(4*n,1); % initialize ODE solution vector
NetFx = zeros(1,n);
NetFy = zeros(1,n);
NetV = 0;

for k=1:NLrows

    i = NL(k,1);
    j = NL(k,2);

    Y(i) = X(i+n); % dr/dt = v in x direction for i atom
    Y(i+2*n) = X(i+3*n); % dr/dt = v in y direction for i atom

    Y(j) = X(j+n); % dr/dt = v in x direction for j atom
    Y(j+2*n) = X(j+3*n); % dr/dt = v in y direction for j atom

    dx = X(i)-X(j);
    dy = X(i+2*n)-X(j+2*n);
    d = CalculateDistance(dx,dy);

    ux = dx/d; % unit vector in x direction
    uy = dy/d; % unit vector in y direction

    Fx = LJForce(d,sigma,epsilon)*ux; % force in x direction
    Fy = LJForce(d,sigma,epsilon)*uy; % force in y direction

    NetV = NetV + 4*epsilon*((sigma/d)^12 - (sigma/d)^6);

    NetFx(i) = NetFx(i) + Fx;
    NetFx(j) = NetFx(j) - Fx;
    NetFy(i) = NetFy(i) + Fy;
    NetFy(j) = NetFy(j) - Fy;

end

for i = 1:n
       Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);
       Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);
end

end
```

LJODE_Fixed.m- ODE function used for the fixed atoms boundary condition. This is used for fixed atoms in both the x and y directions, or fixed in x and fee in y, or vice versa.

```
function[Y] = LJODE_Fixed(t,X,n,m,epsilon,sigma,beta,...
                          Fixed_Atoms,Rc,NLc,tf)

% Construct Neighborlist at first time step
persistent NL NLval

if(t==0)
    NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
    disp('Constructing Neighbor List at t=0');
    NLval = 0;
end

% Neighborlist Reconstruction after defined time interval
if(mod(floor(t),NLc)==0 && floor(t)~=tf)

  if(floor(t) > NLval)
     disp(['Reconstructing Neighbor List at t=', num2str(floor(t))]);
     NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
     NLval = floor(t);
  end

end

[NLrows NLcols] = size(NL);
Y = zeros(4*n,1); % initialize ODE solution vector

NetFx = zeros(1,n);
NetFy = zeros(1,n);

for k=1:NLrows

    i = NL(k,1);
    j = NL(k,2);

    Y(i) = X(i+n); % dr/dt = v in x direction for i atom
    Y(i+2*n) = X(i+3*n); % dr/dt = v in y direction for i atom

    Y(j) = X(j+n); % dr/dt = v in x direction for j atom
    Y(j+2*n) = X(j+3*n); % dr/dt = v in y direction for j atom

    dx = X(i)-X(j);
    dy = X(i+2*n)-X(j+2*n);
    d = CalculateDistance(dx,dy);

    ux = dx/d; % unit vector in x direction
    uy = dy/d; % unit vector in y direction
    Fx = LJForce(d,sigma,epsilon)*ux; % force in x direction
    Fy = LJForce(d,sigma,epsilon)*uy; % force in y direction

    NetFx(i) = NetFx(i) + Fx;
    NetFx(j) = NetFx(j) - Fx;
    NetFy(i) = NetFy(i) + Fy;
    NetFy(j) = NetFy(j) - Fy;

end

for i = 1:n

    if(~any(i==Fixed_Atoms))
        Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);
        Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);
    end

end

end
```

LJODE_Walls.m- ODE function used in walls boundary condition. This is used for walls in both the x and y directions, or for walls in x and free in y or vice versa.

```
function[Y] = LJODE_Walls(t,X,n,m,epsilon,sigma,beta,Rc,NLc,tf,...
                          xbc,ybc,WTM,WBM,WLM,WRM)

persistent NL NLval

if t==0
    NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
    disp('Constructing Neighbor List at t=0');
    NLval = 0;
end

if(mod(floor(t),NLc)==0 && floor(t)~=tf)

  if(floor(t) > NLval)
      disp(['Reconstructing Neighbor List at t=', num2str(floor(t))]);
      NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
      NLval = floor(t);
  end

end

[NLrows NLcols] = size(NL);
Y = zeros(4*n,1); % initialize ODE solution vector

NetFx = zeros(1,n);
```

```
NetFy = zeros(1,n);

for k=1:NLrows

    i = NL(k,1);
    j = NL(k,2);

    Y(i) = X(i+n); % dr/dt = v in x direction for i atom
    Y(i+2*n) = X(i+3*n); % dr/dt = v in y direction for i atom

    Y(j) = X(j+n); % dr/dt = v in x direction for j atom
    Y(j+2*n) = X(j+3*n); % dr/dt = v in y direction for j atom

    dx = X(i)-X(j);
    dy = X(i+2*n)-X(j+2*n);
    d = CalculateDistance(dx,dy);

    ux = dx/d; % unit vector in x direction
    uy = dy/d; % unit vector in y direction

    Fx = LJForce(d,sigma,epsilon)*ux; % force in x direction
    Fy = LJForce(d,sigma,epsilon)*uy; % force in y direction

    NetFx(i) = NetFx(i) + Fx;
    NetFx(j) = NetFx(j) - Fx;
    NetFy(i) = NetFy(i) + Fy;
    NetFy(j) = NetFy(j) - Fy;

end

for i = 1:n

    if(strcmp(xbc,'NoBC')&&strcmp(ybc,'Walls'))
        if(~(X(i+2*n)==WTM && NetFy(i)>0 && X(i+3*n)==0))
            if(~(X(i+2*n)==WBM && NetFy(i)<0 && X(i+3*n)==0))

                Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);

            end
        end

        Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);
    end

    if(strcmp(xbc,'Walls')&&strcmp(ybc,'NoBC'))
        if(~(X(i)==WLM && NetFx(i)<0 && X(i+n)==0))
            if(~(X(i)==WRM && NetFx(i)>0 && X(i+n)==0))

                Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);

            end
        end
        Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);
    end

    if(strcmp(xbc,'Walls')&&strcmp(ybc,'Walls'))
        if(~(X(i+2*n)==WTM && NetFy(i)>0 && X(i+3*n)==0))
            if(~(X(i+2*n)==WBM && NetFy(i)<0 && X(i+3*n)==0))

                Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);

            end
        end
        if(~(X(i)==WLM && NetFx(i)<0 && X(i+n)==0))
            if(~(X(i)==WRM && NetFx(i)>0 && X(i+n)==0))

                Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);

            end
        end
    end

end

end
```

LJODE_Periodic.m- ODE function used for periodic boundaries. This is used for periodic boundaries in both the x and y directions, or periodic in x and free in y, or vice versa.

```
function[Y] = LJODE_Periodic(t,X,n,m,epsilon,sigma,beta,...
                             Lx,Ly,xbc,ybc,Rc,NLc,tf)

persistent NL NLval

if(t==0)
    NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
    disp('Constructing Neighbor List at t=0');
    NLval = 0;
end

% Neighborlist Reconstruction after defined time interval
if(mod(floor(t),NLc)==0 && floor(t)~=tf)

  if(floor(t) > NLval)
      disp(['Reconstructing Neighbor List at t=', num2str(floor(t))]);
      NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
      NLval = floor(t);
  end
```

```
end

[NLrows NLcols] = size(NL);
Y = zeros(4*n,1); % initialize ODE solution vector
NetFx = zeros(1,n);
NetFy = zeros(1,n);

for k=1:NLrows

    i = NL(k,1);
    j = NL(k,2);

    Y(i) = X(i+n); % dr/dt = v in x direction for i atom
    Y(i+2*n) = X(i+3*n); % dr/dt = v in y direction for i atom

    Y(j) = X(j+n); % dr/dt = v in x direction for j atom
    Y(j+2*n) = X(j+3*n); % dr/dt = v in y direction for j atom

    dx = X(i)-X(j);
    if(strcmp(xbc,'Periodic'))
        if(dx >= Lx/2)
            dx = dx - Lx;
        elseif(dx < -Lx/2)
            dx = dx + Lx;
        end
    end

    dy = X(i+2*n) - X(j+2*n);
    if(strcmp(ybc,'Periodic'))
        if(dy >= Ly/2)
            dy = dy - Ly;
        elseif(dy < -Ly/2)
            dy = dy + Ly;
        end
    end

    d = CalculateDistance(dx,dy);

    ux = dx/d; % unit vector in x direction
    uy = dy/d; % unit vector in y direction

    Fx = LJForce(d,sigma,epsilon)*ux; % force in x direction
    Fy = LJForce(d,sigma,epsilon)*uy; % force in y direction

    NetFx(i) = NetFx(i) + Fx;
    NetFx(j) = NetFx(j) - Fx;
    NetFy(i) = NetFy(i) + Fy;
    NetFy(j) = NetFy(j) - Fy;

end
```

```
for i = 1:n

        Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);
        Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);

end

end
```

LJODE_FixedWalls.m- ODE function used in the case of fixed atoms with walls. This is used for fixed atoms in x and walls in y, or vice versa.

```
function[Y] = LJODE_FixedWalls(t,X,n,m,epsilon,sigma,beta,...
                               Fixed_Atoms,Rc,NLc,tf,..
                               xbc,ybc,WTM,WBM,WLM,WRM)

persistent NL NLval

if(t==0)
    NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
    disp('Constructing Neighbor List at t=0');
    NLval = 0;
end

% Neighborlist Reconstruction after defined time interval
if(mod(floor(t),NLc)==0 && floor(t)~=tf)

  if(floor(t) > NLval)
      disp(['Reconstructing Neighbor List at t=', num2str(floor(t))]);
      NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
      NLval = floor(t);
  end

end

[NLrows NLcols] = size(NL);
Y = zeros(4*n,1); % initialize ODE solution vector

NetFx = zeros(1,n);
NetFy = zeros(1,n);

for k=1:NLrows

    i = NL(k,1);
    j = NL(k,2);

    Y(i) = X(i+n); % dr/dt = v in x direction for i atom
    Y(i+2*n) = X(i+3*n); % dr/dt = v in y direction for i atom
    Y(j) = X(j+n); % dr/dt = v in x direction for j atom
    Y(j+2*n) = X(j+3*n); % dr/dt = v in y direction for j atom

    dx = X(i)-X(j);
    dy = X(i+2*n)-X(j+2*n);
    d = CalculateDistance(dx,dy);

    ux = dx/d; % unit vector in x direction
    uy = dy/d; % unit vector in y direction

    Fx = LJForce(d,sigma,epsilon)*ux; % force in x direction
    Fy = LJForce(d,sigma,epsilon)*uy; % force in y direction

    NetFx(i) = NetFx(i) + Fx;
    NetFx(j) = NetFx(j) - Fx;
    NetFy(i) = NetFy(i) + Fy;
    NetFy(j) = NetFy(j) - Fy;

end

for i = 1:n

    if(~any(i==Fixed_Atoms))

        if(strcmp(ybc,'Walls'))
            if(~(X(i+2*n)==WTM && NetFy(i)>0 && X(i+3*n)==0))
               if(~(X(i+2*n)==WBM && NetFy(i)<0 && X(i+3*n)==0))

                  Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);

               end
            end

            Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);
        end

        if(strcmp(xbc,'Walls'))
            if(~(X(i)==WLM && NetFx(i)<0 && X(i+n)==0))
               if(~(X(i)==WRM && NetFx(i)>0 && X(i+n)==0))

                  Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);

               end
            end

            Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);
        end

    end

end
end
```

LJODE_FixedPeriodic.m- ODE function used in for fixed atoms with periodic boundaries. This is used for fixed atoms in x and periodic in y, or vice versa.

```
function[Y] = LJODE_FixedPeriodic(t,X,n,m,epsilon,sigma,beta,...
                                 Fixed_Atoms,Lx,Ly,xbc,ybc,...
                                 Rc,NLc,tf)

persistent NL NLval

if(t==0)
    NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
    disp('Constructing Neighbor List at t=0');
    NLval = 0;
end

if(mod(floor(t),NLc)==0 && floor(t)~=tf)

  if(floor(t) > NLval)
      disp(['Reconstructing Neighbor List at t=', num2str(floor(t))]);
      NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
      NLval = floor(t);
  end

end

[NLrows NLcols] = size(NL);
Y = zeros(4*n,1); % initialize ODE solution vector

NetFx = zeros(1,n);
NetFy = zeros(1,n);

for k=1:NLrows

    i = NL(k,1);
    j = NL(k,2);

    Y(i) = X(i+n); % dr/dt = v in x direction for i atom
    Y(i+2*n) = X(i+3*n); % dr/dt = v in y direction for i atom

    Y(j) = X(j+n); % dr/dt = v in x direction for j atom
    Y(j+2*n) = X(j+3*n); % dr/dt = v in y direction for j atom

    dx = X(i)-X(j);
    if(strcmp(xbc,'Periodic'))
        if(dx >= Lx/2)
            dx = dx - Lx;
        elseif(dx < -Lx/2)
            dx = dx + Lx;
        end
    end

    dy = X(i+2*n)-X(j+2*n);
    if(strcmp(ybc,'Periodic'))
        if(dy >= Ly/2)
            dy = dy - Ly;
        elseif(dy < -Ly/2)
            dy = dy + Ly;
        end
    end

    d = CalculateDistance(dx,dy);

    ux = dx/d; % unit vector in x direction
    uy = dy/d; % unit vector in y direction

    Fx = LJForce(d,sigma,epsilon)*ux; % force in x direction
    Fy = LJForce(d,sigma,epsilon)*uy; % force in y direction

    NetFx(i) = NetFx(i) + Fx;
    NetFx(j) = NetFx(j) - Fx;
    NetFy(i) = NetFy(i) + Fy;
    NetFy(j) = NetFy(j) - Fy;

end

for i = 1:n

    if(~any(i==Fixed_Atoms))
        Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);
        Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);
    end

end

end
```

LJODE_WallsPeriodic.m- ODE function used for walls with periodic boundaries. This is used for walls in x and periodic boundaries in y, or vice versa.

```
function[Y] = LJODE_WallsPeriodic(t,X,n,m,epsilon,sigma,beta,...
                                  Lx,Ly,xbc,ybc,Rc,NLc,tf,...
                                  WTM,WBM,WLM,WRM)

% Construct Neighborlist at first time step
persistent NL NLval

if(t==0)
```

```
    NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
    disp('Constructing Neighbor List at t=0');
    NLval = 0;
end

% Neighborlist Reconstruction after defined time interval
if(mod(floor(t),NLc)==0 && floor(t)~=tf)

  if(floor(t) > NLval)
      disp(['Reconstructing Neighbor List at t=', num2str(floor(t))]);
      NL = neighborList(X(1:n), X(2*n+1:3*n), Rc, n);
      NLval = floor(t);
  end

end

[NLrows NLcols] = size(NL);
Y = zeros(4*n,1); % initialize ODE solution vector

NetFx = zeros(1,n);
NetFy = zeros(1,n);

for k=1:NLrows

    i = NL(k,1);
    j = NL(k,2);

    Y(i) = X(i+n); % dr/dt = v in x direction for i atom
    Y(i+2*n) = X(i+3*n); % dr/dt = v in y direction for i atom

    Y(j) = X(j+n); % dr/dt = v in x direction for j atom
    Y(j+2*n) = X(j+3*n); % dr/dt = v in y direction for j atom

    dx = X(i) - X(j);
    if(strcmp(xbc,'Periodic'))
        if(dx >= Lx/2)
            dx = dx - Lx;
        elseif(dx < -Lx/2)
            dx = dx + Lx;
        end
    end

    dy = X(i+2*n) - X(j+2*n);
    if(strcmp(ybc,'Periodic'))
        if(dy >= Ly/2)
            dy = dy - Ly;
        elseif(dy < -Ly/2)
            dy = dy + Ly;
        end
    end

    d = CalculateDistance(dx,dy);
```

```
        ux = dx/d; % unit vector in x direction
        uy = dy/d; % unit vector in y direction

        Fx = LJForce(d,sigma,epsilon)*ux; % force in x direction
        Fy = LJForce(d,sigma,epsilon)*uy; % force in y direction

        NetFx(i) = NetFx(i) + Fx;
        NetFx(j) = NetFx(j) - Fx;
        NetFy(i) = NetFy(i) + Fy;
        NetFy(j) = NetFy(j) - Fy;

    end

    for i = 1:n

        if(strcmp(ybc,'Walls'))
            if(~(X(i+2*n)==WTM && NetFy(i)>0 && X(i+3*n)==0))
                if(~(X(i+2*n)==WBM && NetFy(i)<0 && X(i+3*n)==0))

                    Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);

                end
            end

            Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);
        end

        if(strcmp(xbc,'Walls'))
            if(~(X(i)==WLM && NetFx(i)<0 && X(i+n)==0))
                if(~(X(i)==WRM && NetFx(i)>0 && X(i+n)==0))

                    Y(i+n) = Y(i+n) + NetFx(i)/m - beta/m*X(i+n);

                end
            end

            Y(i+3*n) = Y(i+3*n) + NetFy(i)/m - beta/m*X(i+3*n);
        end


    end

    end
```

LJForce.m- Function used to calculate the force between a pairwise group of atoms given the distance between them. The function recieves the distance between atoms as an input, and returns the force between these atoms according to Lennard-Jones.

```
function [LJF] = LJForce(d, sigma, epsilon)

LJF = epsilon*(48*sigma^12/d^13 -24*sigma^6/d^7);
```

CalculateDistance.m- Function used to calculate the distance between two atoms in two dimensional space. The funcion recieves the x component of the distance between two atoms, *dx* and the y component distance, *dy*. The return value is the distance between two atoms.

```
function [d] = CalculateDistance(dx,dy)

d = sqrt(dx^2+dy^2);
```

---

neighborList.m- Function used to construct a neighbor list of atoms in two dimensions. The function recieves the cutoff radius as an input and constructs the neighbor list accordingly. The return value is a two column vector that stores pairs of atoms that are within the scope of the cutoff radius.

```
function [NL] = neighborList(xx, xy, Rc, n)

NL = zeros(1,2);

for i=1:n
    for j = i+1:n

        dx = xx(i) - xx(j);
        dy = xy(i) - xx(j);
        d = CalculateDistance(dx,dy);

        if( d < Rc && i~=j)
            NL(end+1,:) = [i, j];
        end
    end
end

NL = NL(2:end,:);
```

---

timestep.m- Function that is used to calculate the next time step when ode45 breaks out of solving due to walls or periodic boundaries.

```
function [ts,Nnew] = timestep(TE,ttf,N,dN)

f = floor(TE);
c = ceil(dN*(TE-f));
```

```
ts = c/dN+f;

Nnew = N-numel(ttf)+1;
```